USER MANUAL

Seneca Z-TWS3 and Z-TWS64

This document is property of SENECA srl. Duplication and reproduction of its are forbidden (though partial), if not authorized. Contents of present documentation refers to products and technologies described in it. Though we strive for reach perfection continually, all technical data contained in this document may be modified or added due to technical and commercial needs; it's impossible eliminate mismatches and discordances completely. Contents of present documentation is anyhow subjected to periodical revision. If you have any questions don't hesitate to contact our structure or to write us to e-mail addresses as above mentioned.

(01943) 602001 @ sales@issltd.co.uk www.issltd.co.uk

SUMMARY

Systems based on Seneca Z-PC Line: 6
Z-TWS and I/O modules 6
PLC FOR SMALL AND MEDIUM AUTOMATIONS 6
SUPERVISORY OF PLANTS AND DATA ACQUISITION 7
ARCHITECTURE 8
Z-TWS: A TINY WEB SERVER 8
I/O MODBUS AND I/O CANopen 9
DATA ACQUISITION MODULE (DAM) 9
WEB SERVER 9
PLC CODESYS 9
RETENTIVE VARIABLES (FeRAM) 11
WEB EDITOR 13
DATA RECORDER 14
TREND VIEWER 15
Z-NET AND TREND VIEWER INSTALLATION 16
Introduction of CoDeSys and its installation 18
WHAT IS CODESYS? 18
HOW TO INSTALL CODESYS 18
HOW TO INCLUDE THE SENECA LIBRARIES INTO CODESYS 19
Appendix A: About the PLC.ini file 66
Appendix B: Seneca libraries for CoDeSys 70
Appendix C: CANopen I/O with ISAGRAF PLC 132
Appendix D: Library "Controlli" for Microsoft® Visual Studio .NET 135

CPU UNIT		
Features	**Z-TWS3**	**Z-TWS64**
COMMUNICATION PORT		
SERIAL PORT nr.0 RS232 (2400 bps, 8N1)	Yes	Yes
SERIAL PORT nr.1 MODBUS/CAN (1200..115200 bps Stop bit, parity)	Yes (or rear from IDC10)	Yes (or rear from IDC10)
SERIAL PORT nr.2 RS232/RS485	Yes	Yes
SERIAL PORT nr.3 RS485	Yes	Yes
ETHERNET COMMUNICATION		
Ethernet port number	1	1
Ethernet TCP/IP	Yes	Yes
Modbus TCP/IP	Client/server	Client/server
Speed (Ethernet)	10 Mb/s	100 Mb/s
SERIAL/BUS COMMUNICATION		
Max task modbus master	4	4
Max task modbus slave RTU or TCP	3	3
Modbus RTU on RS232/RS485	Master/slave	Master/slave
Speed (serial)	1200..115200 bps	1200..115200 bps
Parity	None, even, odd	None, even, odd
Max distance connection	1200 m	1200 m
CANopen master	Yes	Yes
CPU & MEMORY		
CPU	SCM 20 - RISC	SCM 40 - RISC
Memory	8 Mbyte of RAM 16 Mbyte of flash	64 Mbyte of RAM 128 Mbyte of flash
Retentive variables	8 kb	8 kb
Battery back-up	yes	Yes
CONNECTIONS		
	Removable 3-way screw terminals, 5.08 mm pitch	Removable 3-way screw terminals, 5.08 mm pitch
	Rear IDC10 connector for DIN 46277 rail	Rear IDC10 connector for DIN 46277 rail
	RJ10-4/4, RJ45 (on frontal)	RJ10-4/4, RJ45 (on frontal)
SYSTEM PROTOCOLS		
Tunneling VPN	Yes	Yes
PPP	Yes	Yes
HTTP	Yes	Yes
FTP	Yes	Yes
SMTP	Yes	Yes
CONFIGURATIONS, SOFTWARE, STANDARDS		
Z-NET	Yes	Yes
OPC server	Yes	Yes

Web-server	Yes	Yes
Datalogger	Yes	Yes
PLC/CoDeSys	Yes	Yes
Standards	EN50081-2; EN55011; EN50082-2; EN61000-2-2/4; EN50140/141; EN61010-1; EN60742	
POWER SUPPLY		
Supply voltage	10-40 Vdc or 19-28 Vac (50-60 Hz)	
Power consumption	Max 3.5 W	Max 4 W

Jumper setting

A correct position of jumpers is necessary to choose a specified combination of the various communication ports. An incorrect configurations preclude the employment of the module.

See installation manual for more informations.

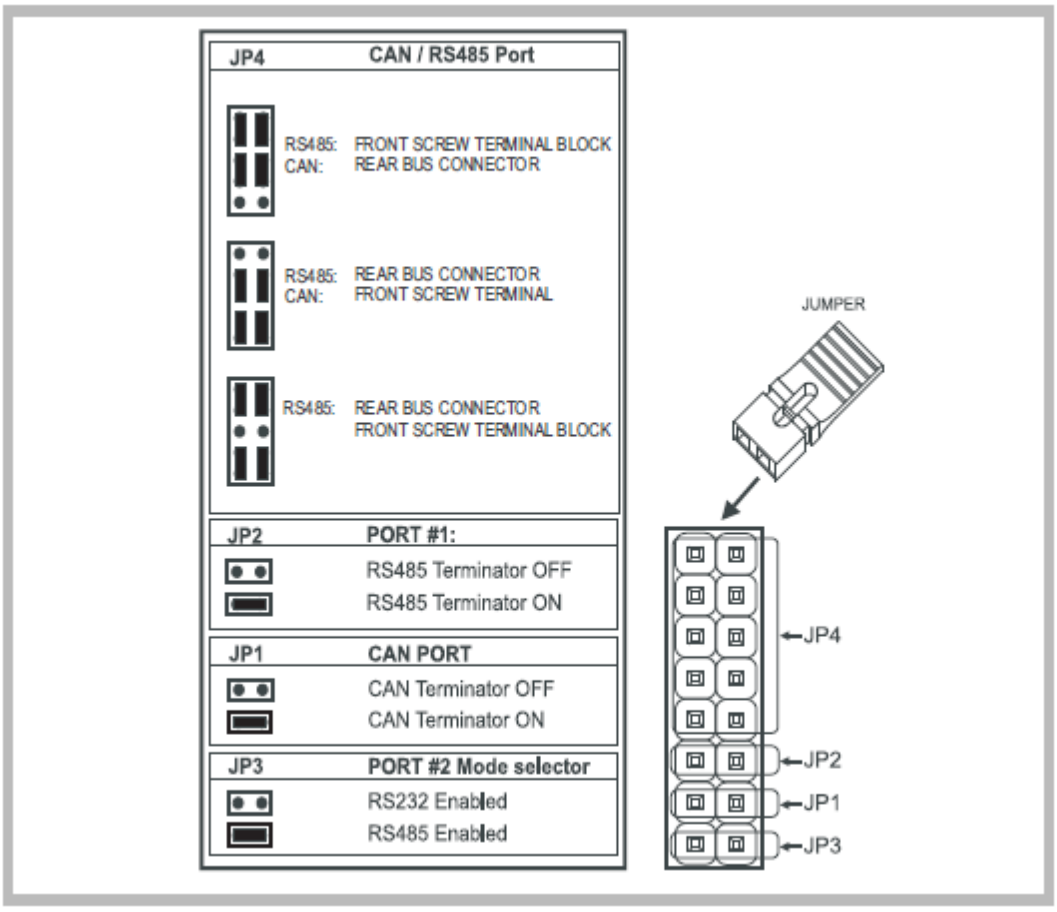


Figure 1 - Jumper setting

Autodiagnostic Functions

The Z-TWS is equipped with four LEDs on the front, one of which indicates the status of form.
At startup, the Z-TWS performs some internal controls (lasting a few tens of seconds) and loads the operating system, after which the LED 'Run' will be piloted in the following way:
LED Off = PLC STOP
LED On = PLC in RUN
Flashing LED = fault PLC
If the LED is flashing, it will be necessary to carry out checks for appropriate resolve the anomaly.

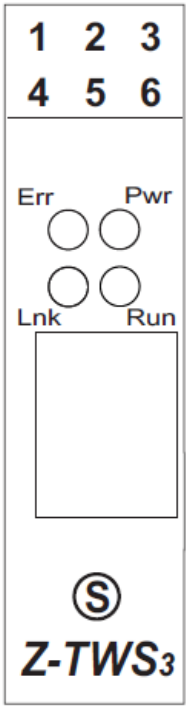


Figure 2 - Front panel of Z-TWS module

Systems based on Seneca Z-PC Line: Z-TWS and I/O modules

The Z-TWS module is a device that combines the functions of PLC, web server, data logger, data acquisition; thanks to its flexibility of configuration can be used in many configurations and architectures different from each other.

The combination of data acquisition process, PLC module and WEB SERVER makes it possible to use the module as a Z-TWS datalogger – Trend register: the signals from field and possible PLC variables are stored at predetermined intervals. The WEB SERVER through Web pages (properly programmed; see the Web Editor chapter) offers the possibility to display the time trend of signals, as well as the possibility to download these values in a Microsoft® Excel® format. Complex computations of signals are possible during acquisition, then stored.

The stored data can also be acquired through the supervisory systems, as SCADA. The Z-TWS allows two standard modalities for the acquisition of variables to a remote monitoring system: OPC Server and Modbus TCP IP.

Fields of use of the Z-TWS are represented by all small/medium-sized automation applications, which may require significant capacity calculation, for industrial process control and remote control.

A system based on Seneca Z-PC Line (Z-TWS unit with I/O modules) has three characteristics:

1) it is a **distributed system**: I/O is acquired through RS485 bus-communication or CANopen bus communication

2) it is a **open system**: I/O data acquisition protocol (Modbus RTU), Ethernet interface protocol (HTTP, FTP, Modbus TCP) and CANopen are standard protocols

3) it is an **integrated system**: all the functions needed for specific applications are enclosed in a single device and are configurable via software

Main functions and features of Z-TWS device are described in the following lines.

PLC FOR SMALL AND MEDIUM AUTOMATIONS

The Z-TWS is a PLC (programmable logic controller) used to manage small/medium dimension plants. It is optimized to process a large number of data and to perform analogue controls: the Z-TWS processes floating point calculations and it consists of a significant amount of memory.

The system used for field connection and data acquisition is based on a RS485 bus with Modbus-RTU protocol or a CAN bus with CANopen protocol. The Seneca Z-PC Line is constituted by digital and analogue modules for I/O management.

The Z-TWS has a RS232 serial port, for example, to connect the Z-TWS to a PC or modem using RS232, to display diagnostics of functioning.

(01943) 602001 @ sales@issltd.co.uk www.issltd.co.uk

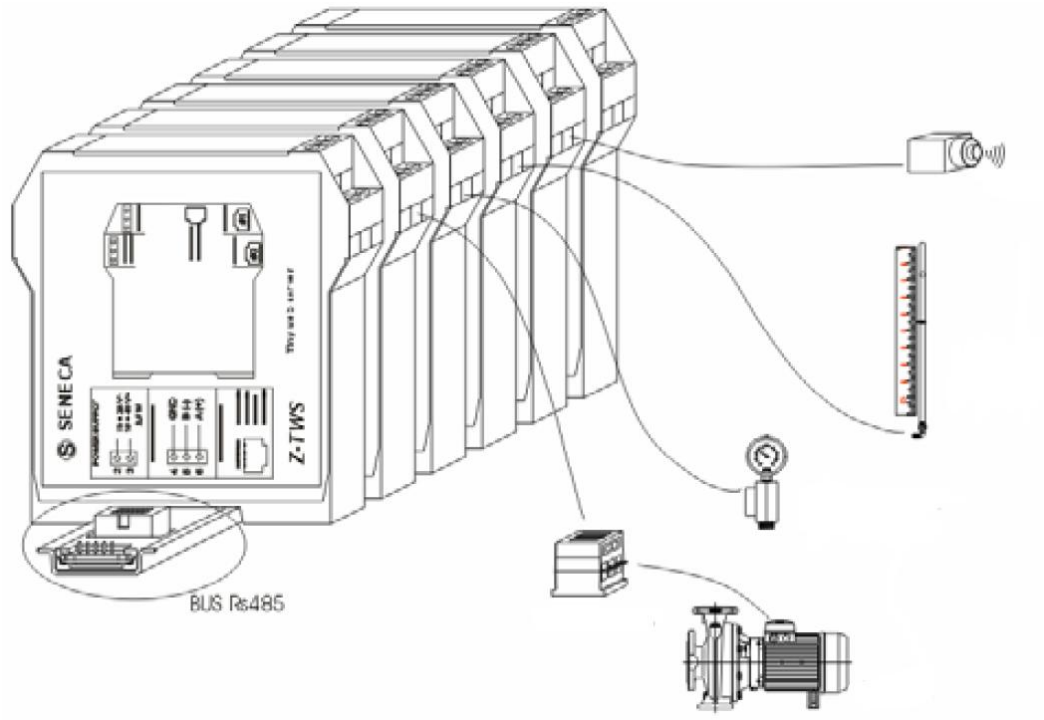


Figure 3 - TWS used to manage an automation plants

SUPERVISORY OF PLANTS AND DATA ACQUISITION

Thanks to the Ethernet TCP / IP interface, it is possible to connect the Z-TWS device to supervisory and/or data acquisition stations. The following communication protocols are implemented:

HTTP

Advanced communication that allows a complete control of the device: reading and writing of variables, file transfer in reading/writing, complete control of the PLC and data acquisition (start, stop, restart), IP configuration parameters.

FTP

Service protocol used to transfer configuration files or web pages.

MODBUS/TCP

Standard protocol used to capture and write all I/O variables of PLC.

SMTP

This protocol is implemented in a complete modality (selection of addresses, server, object, text, etc…) and it allows the e-mail sending.

ARCHITECTURE

Z-TWS: A TINY WEB SERVER

The Z-TWS device is based on the most widespread standards (Ethernet interface, stack TCP/IP, HTTP server) and allows the integration of any systems based on Modbus RTU protocol and serial interface RS485 with Intranet/Internet network.

When the HTML pages are uploaded (see Z-NET web editor software), they can be downloaded into the flash disk, using FTP protocol. The module can be protected by password, to avoid unauthorized access.

Moreover, if you connect the Z-TWS to a GSM or GPRS modem, it is possible to send related SMS or call the interested center.

The following figure shows the architecture of TWS with I/O serial modules, the related interfaces and the communication between them.

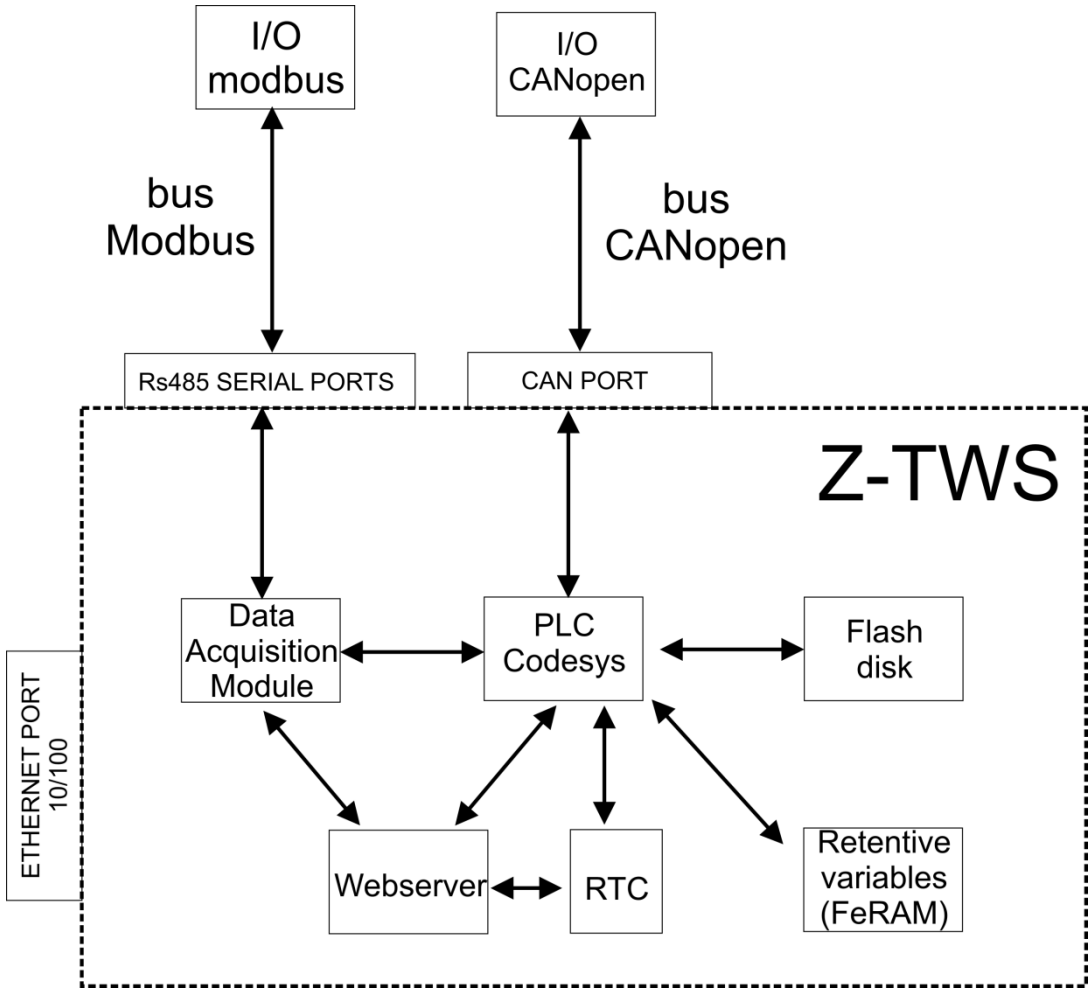


Figure 4 - Block diagram of a Z-TWS module

I/O MODBUS AND I/O CANopen

The block “I/O Modbus” means one or more modules that are connected with the Z-TWS through a serial Modbus bus (for example: Seneca Z-4TC)

The block “I/O CANopen” means one or more modules that are connected with the Z-TWS through a CANopen bus (for example: Seneca ZC-8AI).

DATA ACQUISITION MODULE (DAM)

The thread that runs with highest priority in the system is the acquisition of new data from I/O Modbus module to DAM. In particular, the writing phase is not managed by DAM, but in an asynchronous way.

For example: if the Z-TWS have to read the content of registers: 40022, 40023, 40025, 40027, the DAM allows to optimize the reading phase, because it is not necessary to perform six readings (registers from 40022 to 40027), but only one.

DAM needs to be considered in PLC cycles based on Modbus protocol, but doesn’t need to be considered in PLC cycles based on CANopen protocol (the data are acquired from program directly).

A file called **PLC.ini** is sent to the Data Acquisition Module (DAM) and contains the modalities to read some Modbus registers, for example (example: read the registers from 40120 to 40128, etc…). The PLC.ini file is easily configurable by the free Z-NET software.

The MODBUS protocol is initialized with the parameters normally contained in this file that must be stored in the root directory of the flash-drive Z-TWS.

The PLC.ini file contains the parameters for all the four serial ports.

More information about the file PLC.ini are described in the appendix A.

WEB SERVER

The Z-TWS has a built-in webserver, so by Z-NET Web Editor it is possible to create and upload the HTML pages: this allows to accede to plants through a remote connection for supervisory, remote control functions and remote support (recovery, variation of parameters, system maintenance, etc…). All system configurations can be performed by remote (TWS configuration, I/O modules, PLC).

PLC CODESYS

The Z-TWS can be used as an integrated PLC, able to manage small/medium sized automation plants, because it consists of a significant amount of memory and computing capacity.

The inputs and the outputs of PLC are modules that can be interfaced with Modbus-RTU protocol or CANopen protocol (like Seneca Z-PC Line module): so, it operates like any control system but the I/O devices are de-centralized

The PLC target is based on CoDeSys® IEC61131 standard; the programming languages supported are:

• contacts (LD);

• flow chart (FC);

• diagram block (FDB);

• sequential flow chart (SFC);

• structured language (ST);

• instruction list (IL);

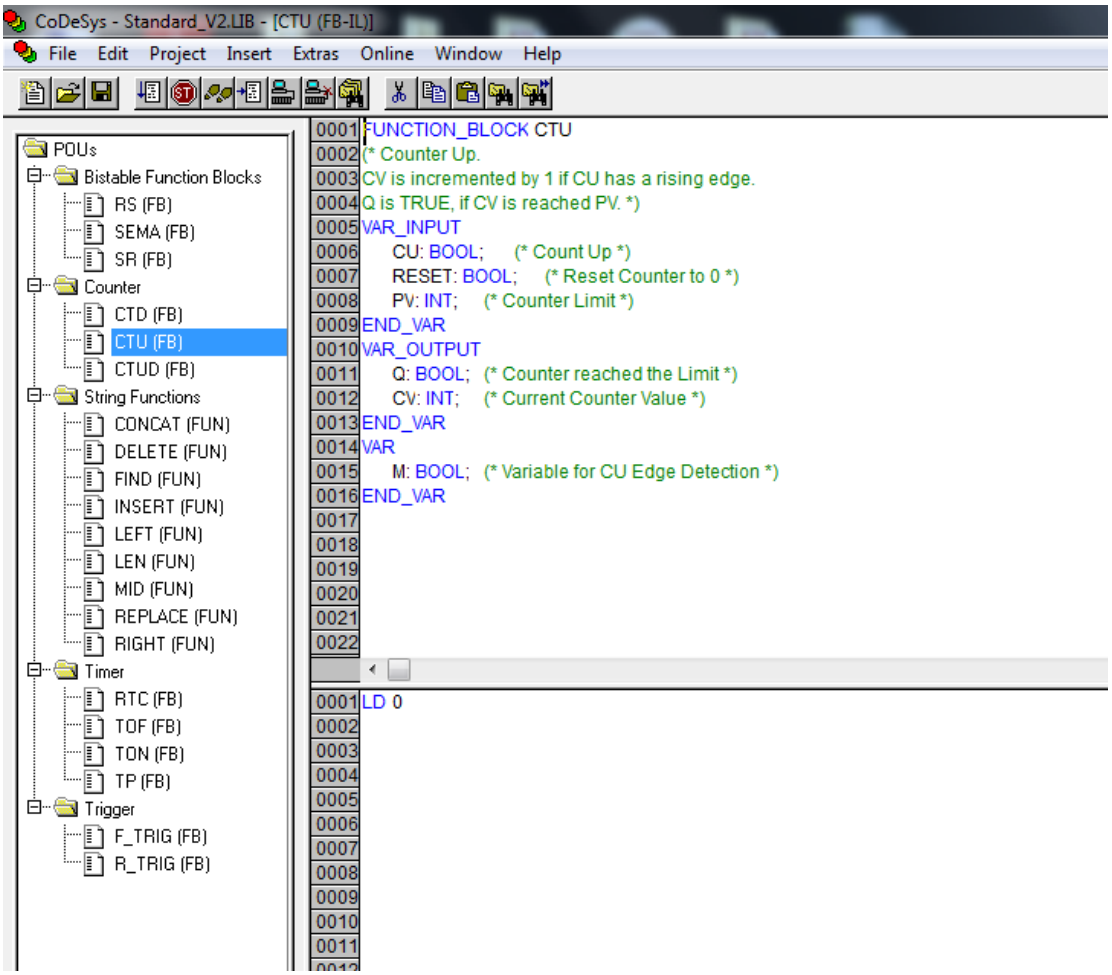


Figure 5 - A CoDeSys ® page

RTC (INTEGRATED MANAGEMENT FOR LOCAL HOUR)

The Z-TWS has an internal hardware clock with a battery that can operate for a month without power.

During the first start or after a prolonged absence of powering (more than one month) will be

reset by the clock at midnight of January 1, 2000.
When restarted, the DAM reads the date and time from this clock and calculates the corresponding UTC respecting the settings for time zone and daylight saving time (DST).
UTC is used as a time stamp of sampling for trend, while the local time (the value read by the clock) is used for all other needs such as event logs or time stamp for the file system.
If the Z-TWS is running when a DST transition occurs (ie the beginning or the end of the period when DST is in force), the clock will be updated automatically. If the transition from summer to winter time or vice versa occurs when the module is off, the clock must be adjusted manually
Time management is carried out according to international standard.

Note: The 'Coordinated Universal Time (UTC)' is the universal standard time.
See website for more information http://wwwghcc.msfc.nasa.gov/utc.html

FLASH DISK

To improve the system performances, it is important to consider that the access to files affects heavily in the cycle time of PLC. So, it is necessary to optimize the accesses to the file system in the PLC program.

The Z-TWS includes a 128 Mbytes (16 Mbytes for Z-TWS3) flash disk that can be used to store files (for example log files), and it is based on FAT file system.

IMPORTANT!

The file system are managed by the OS File System, the number of files that can be opened at the same time is limited, as well the number of files that can be saved in a same directory and in the root.

It is important to consider that on a PLC the flash has a limited cycles of writing.

RETENTIVE VARIABLES (FeRAM)

The Z-TWS is equipped by a 8 kb static RAM (feRAM); into this memory is possible to store data, like a classic RAM memory but with the retentive ability of a flash memory.

An introduction of the Z-NET

The Z-NET is the basic software for the design and configuration of systems Z-TWS based on Modbus I/O; it is a free-downloadable software, and can be downloaded from the website www.seneca.it.

Moreover, the Z-NET software can be installed by the CD “Seneca Package” or freely downloaded from www.seneca.it.

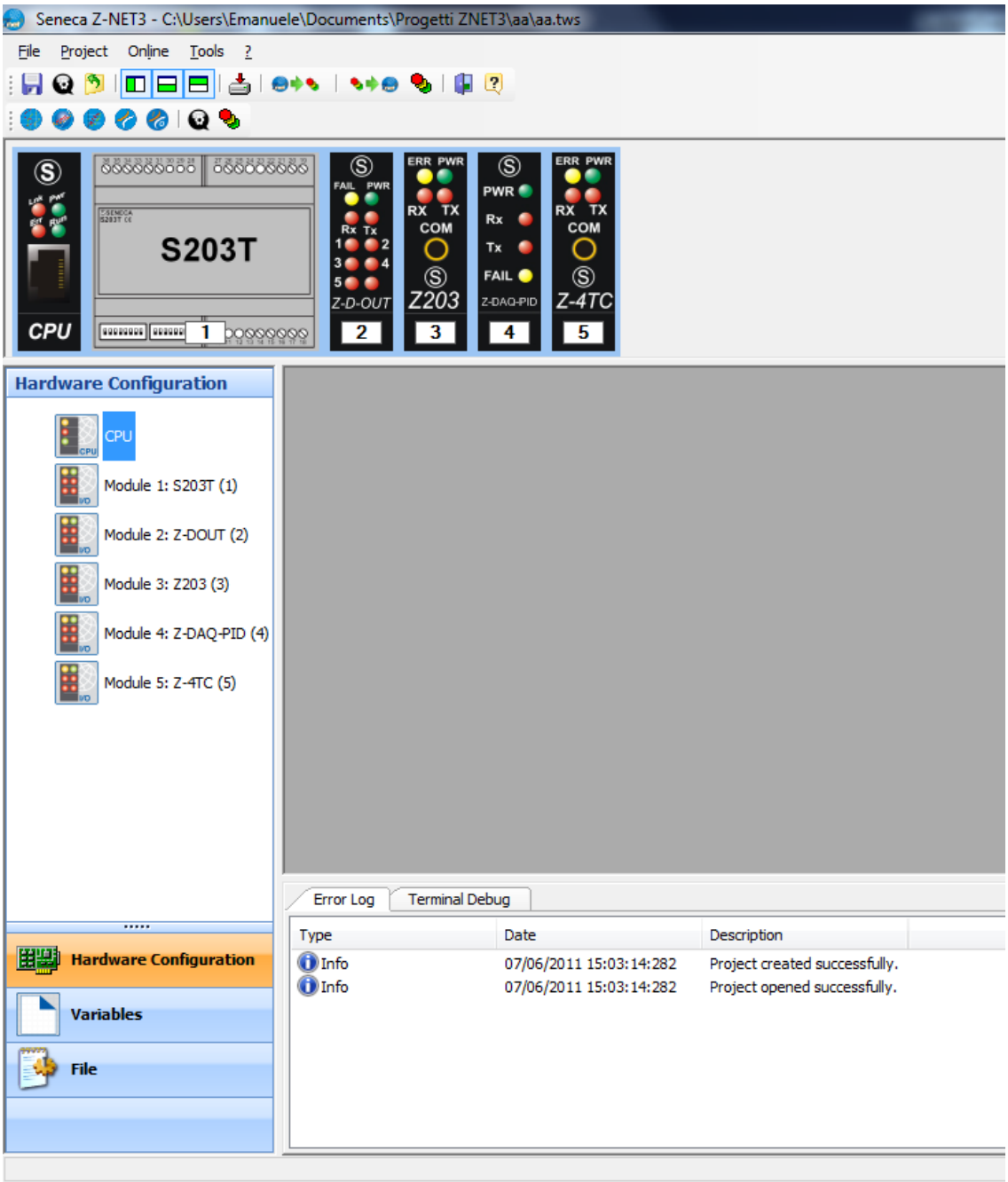


Figure 6 - A Seneca Z-NET page

The Z-NET software allows, easily,

- **to configure a bus with Z-TWS and I/O Modbus**

The Z-NET permits to configure all parameters to manage a bus constitued by a Modbus master, and one or more Modbus slaves.

- **to configure Modbus I/O projects**

The Z-NET permits to configure all parameters to manage a bus constitued by one o more Modbus slaves, with a generic Modbus master.

For these purposes Seneca offers many modules, that can be used depending on the particular demand of the Customers.

- **to configure the Data Recorder and analyze the data**

Data recorder is included a Z-NET that allows to perform the data-logging for all the variables that are available for each modules, and particular operations can be applied to them (alarm, math functions, etc…)

- **to create and configure dynamic HTML pages for the Web Server Z-TWS**

This function is implemented by Web Editor, included into the Z-NET package.

- **to export the hardware configuration for CoDeSys**

The software Z-NET permits to export the configuration to CoDeSys PLC, (**Co**ntroller **De**velopment **Sys**tem) that's a comprehensive software tool for industrial automation technology.

WEB EDITOR

The Web editor software is integrated into the Z-NET: it is an effective tool and it allows to optimize the web server functionality.

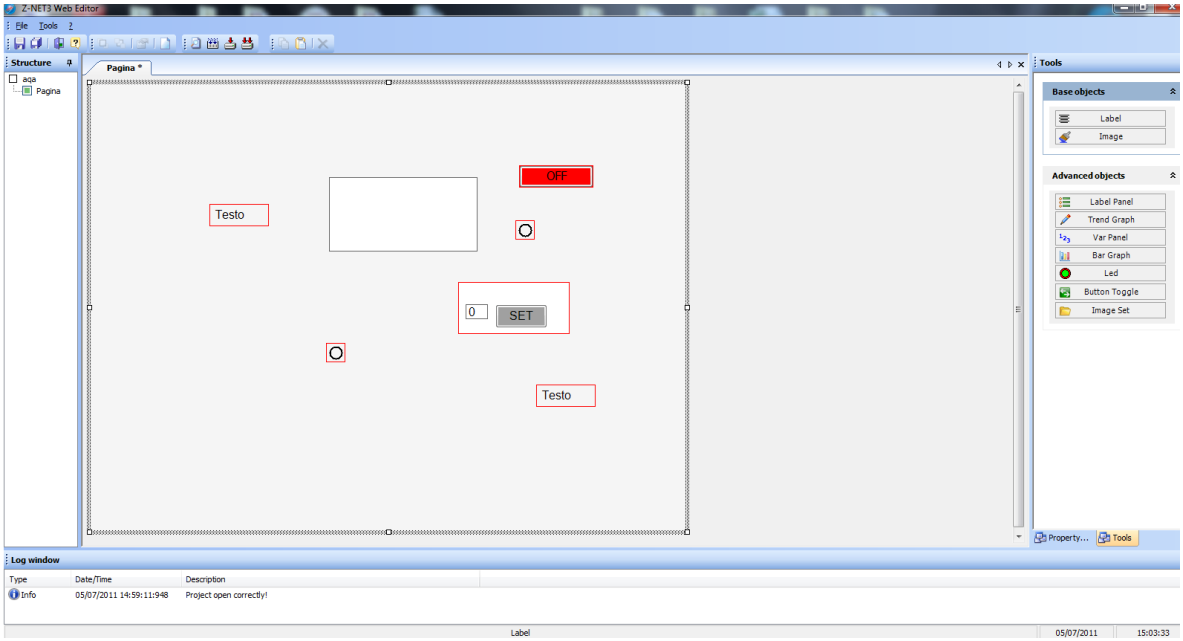


Figure 7 - A Seneca Web Editor page

This software allows:

- **to create easily web pages that contain the applets, developed for Z-TWS**
- **to associate directly to the objects the variables defined in the corresponding Z-NET project**
- **to edit easily the features of the various objects**
- **to download the applets and web pages directly in the CPU**
- **to display the preview of the web pages**

DATA RECORDER

Data recorder is a software that allows to perform the data-logging for each variables.

Starting from a configuration inherited from a project-Z NET, Data Recorder is a software that allows to display and record the variable values of the I/O modules, both through serial (system with I/O devices only) and through Ethernet (Z-TWS+I/O modules).

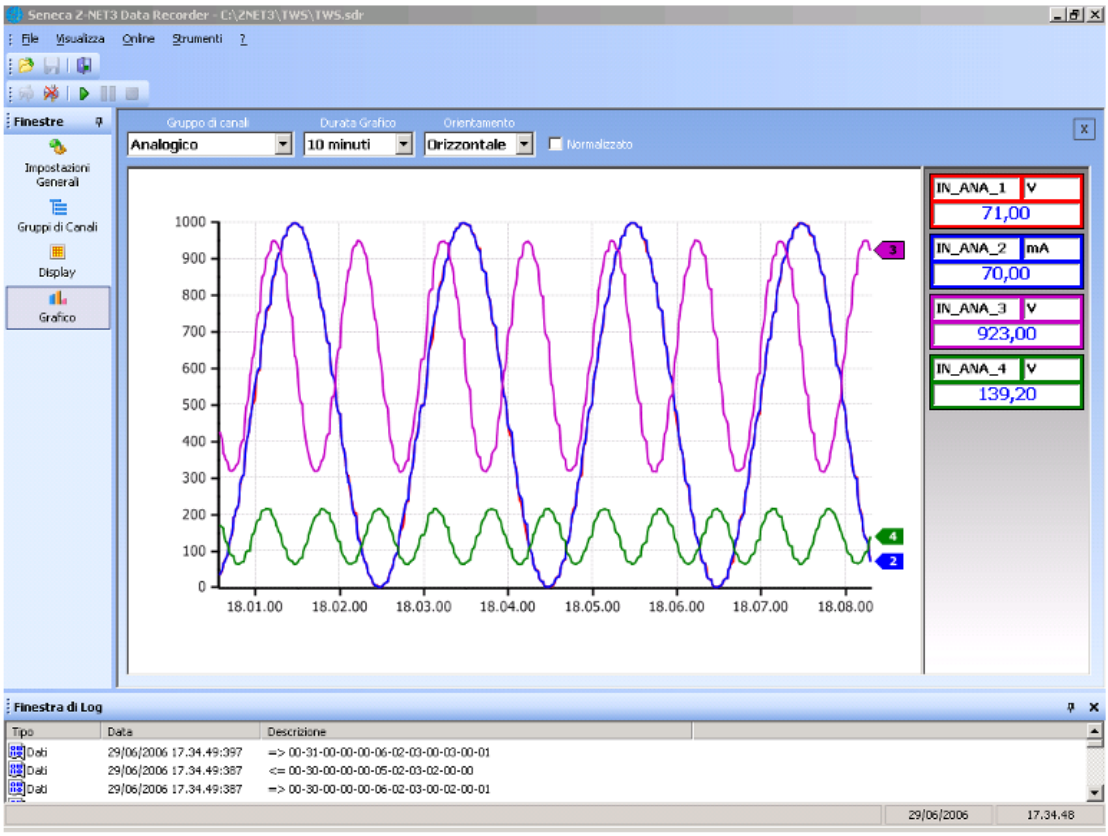


Figure 8 - A Seneca Data Recorder page

This software allows:

- **To define groups of channels that will be defined in a same page**
- **To display the instantaneous value of channels**
- **To display by a graphics the instantaneous value of channels**
- **To display the alarms related to the channels, in real time**
- **To record the values of the channels in an Access database (with an extension .mdb)**
- **To record the alarm events**

Then, using the Trendviewer software, it's possible to view the historical data acquired from the Data Recorder. It is important to emphasize that **a full use of the Data Recorder software requires the purchase of a license (hardware key USB). Contact Seneca for more informations and to acquire a license.** There are several types of licenses, which differ from each other for the maximum number of channels that can be managed.

TREND VIEWER

The Trend Viewer software is a free tool able to view the historical data, recorded by Data Recorder software. When the database .mdb is open (this database contains the recorded data), the following operations are possible:

- **extremely accurate graphic visualization of the historical data, stored by the Data Recorder software**
- **visualization of the alarms, defined and recorded using the datarecorder software**
- **print of the obtained graphics**
- **export of the database currently open in a format CSV**

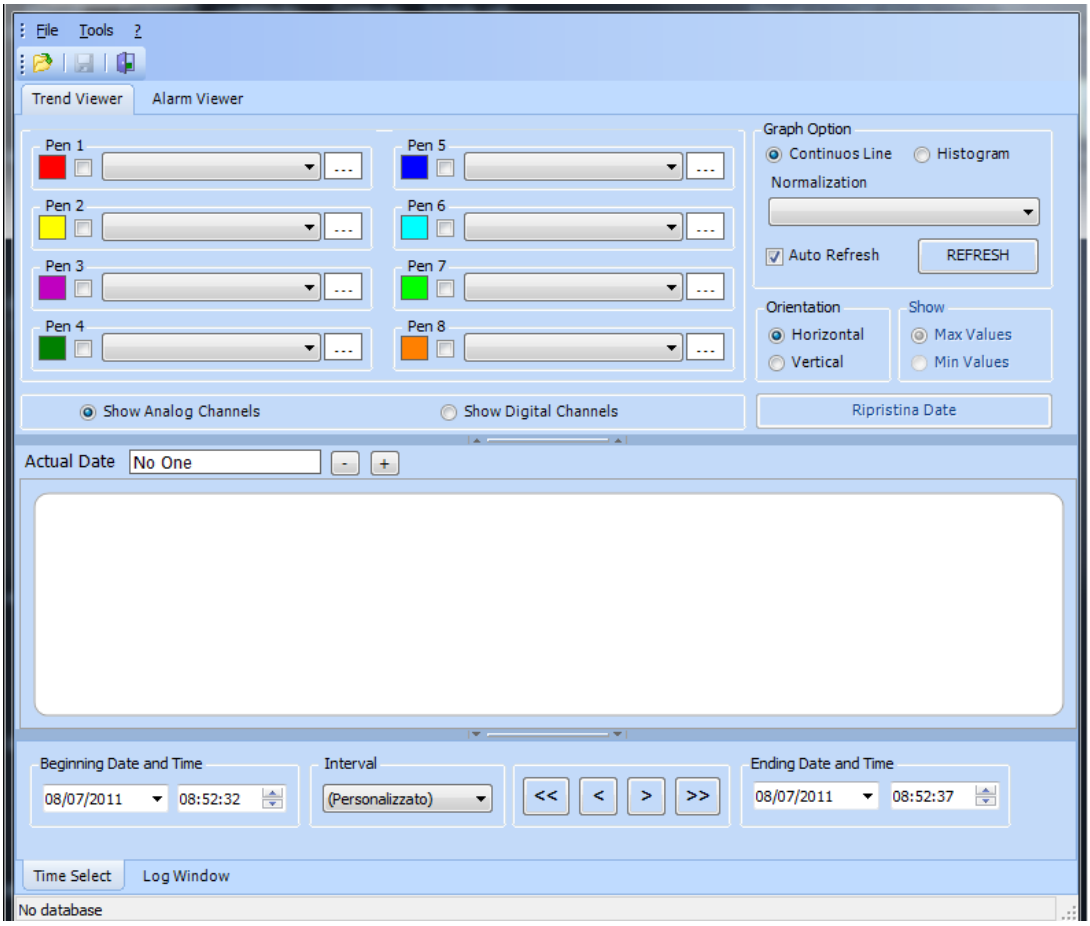


Figure 9 - A Seneca Trend Viewer page

Z-NET AND TREND VIEWER INSTALLATION

Z-TWS is equipped with “Seneca package CD”; with this CD is possible to install the Z-NET software and also TREND VIEWER.

To install the Z-NET software, insert the CD, double click on the file “CD_Start.exe” and so appears the following window.

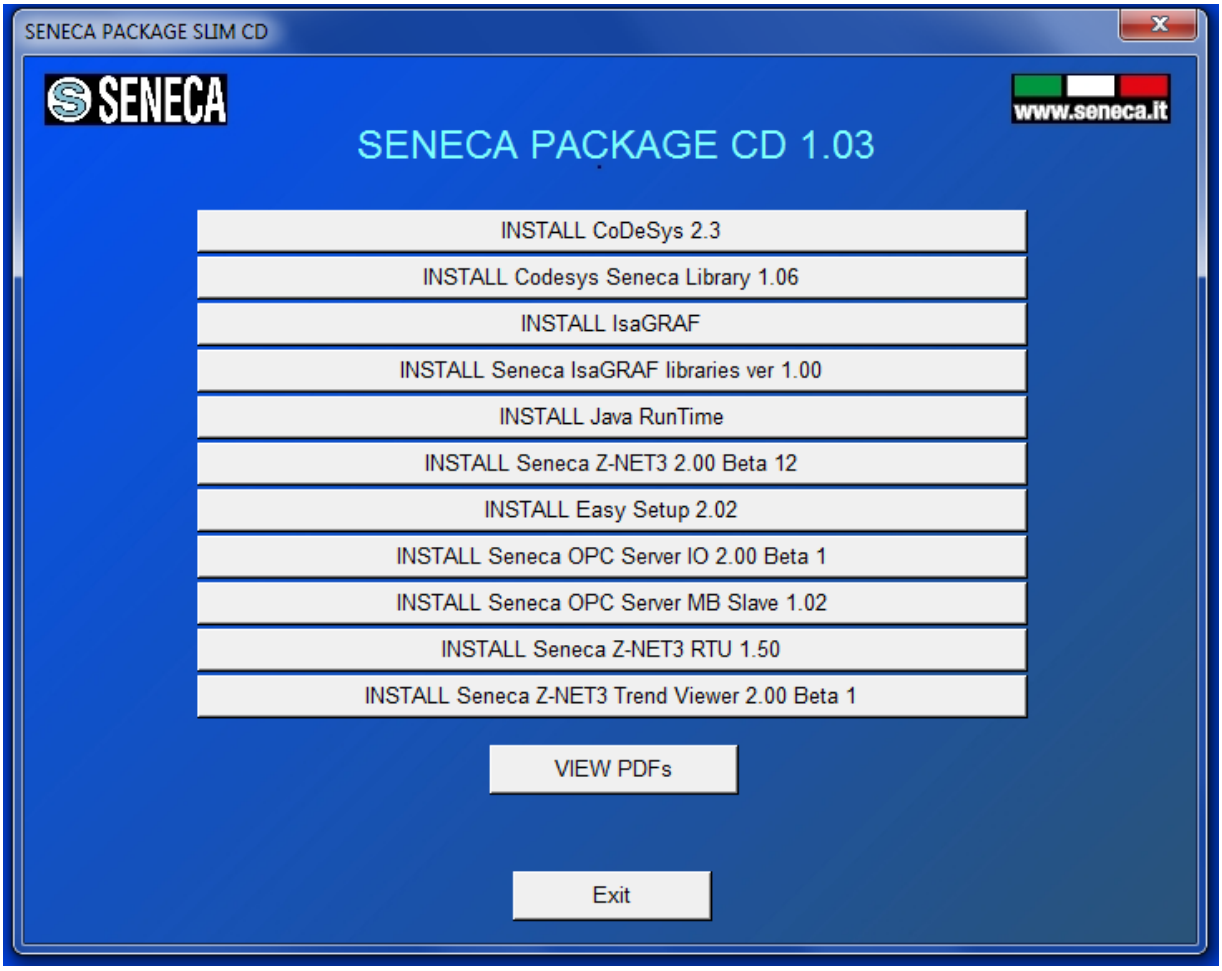

SENECA PACKAGE SLIM CD
SENECA
www.seneca.it
SENECA PACKAGE CD 1.03
INSTALL CoDeSys 2.3
INSTALL Codesys Seneca Library 1.06
INSTALL IsaGRAF
INSTALL Seneca IsaGRAF libraries ver 1.00
INSTALL Java RunTime
INSTALL Seneca Z-NET3 2.00 Beta 12
INSTALL Easy Setup 2.02
INSTALL Seneca OPC Server IO 2.00 Beta 1
INSTALL Seneca OPC Server MB Slave 1.02
INSTALL Seneca Z-NET3 RTU 1.50
INSTALL Seneca Z-NET3 Trend Viewer 2.00 Beta 1
VIEW PDFs
Exit

Figure 10 - The installation page of the Seneca package CD

At the first time, click on “install Seneca Z-NET n.n Beta x.x” to install the Z-NET and follow the simple operations; then click on “install Seneca Z-NET Trend Viewer n.n Beta x.x” to install the Trend Viewer software and follow the simple operations.

Note that Seneca Z-NET contains Data Recorder and Web Editor softwares, while the Trend Viewer have to be installed stand-alone previously; it need to be started after that the file .mdb has been generated by Data Recorder.

(01943) 602001 @ sales@issltd.co.uk www.issltd.co.uk

Introduction of CoDeSys and its installation

WHAT IS CODESYS?

CoDeSys is a complete development environment for your PLC (CoDeSys stands for Controlled Development System).

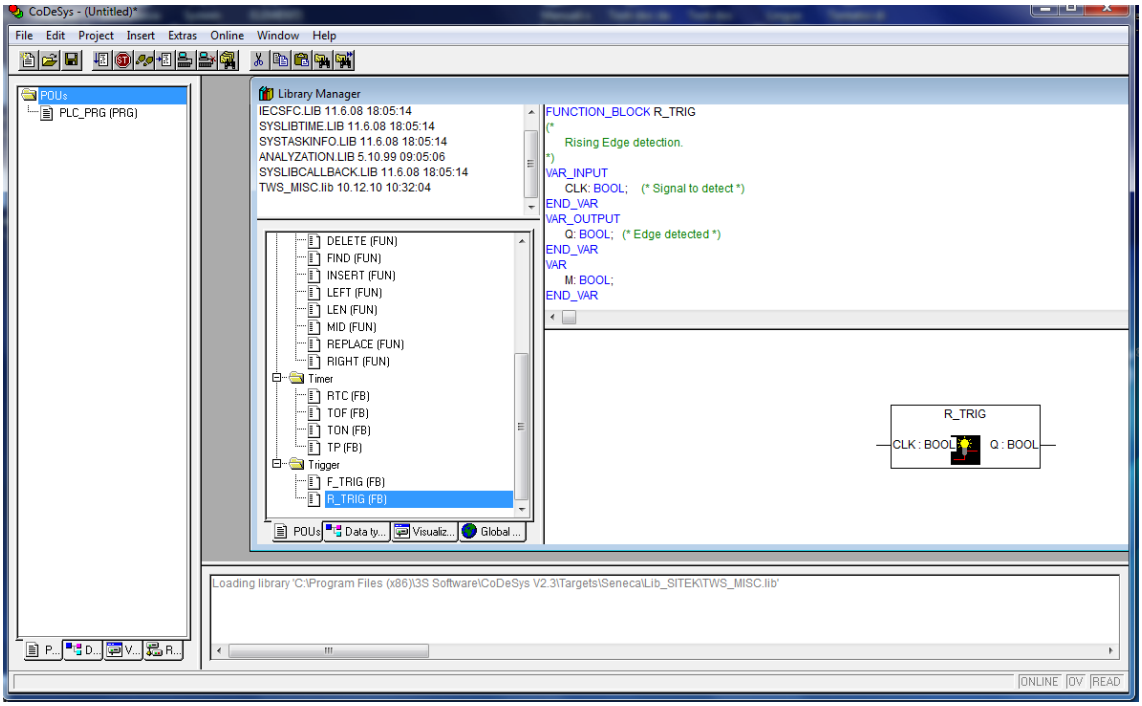


Figure 11 - A Codesys page

CoDeSys puts a simple approach to the powerful IEC language at the disposal of the PLC programmer. Use of the editors and debugging functions is based upon the proven development program environments of advanced programming languages.

HOW TO INSTALL CODESYS

Z-TWS is equipped with “Seneca package CD”; with this CD is possible to install the Codesys software and its Seneca libraries.

To install the Codesys software, insert the CD, double click on the file “CD_Start.exe” and so appears the following window

SENECA PACKAGE SLIM CD
SENECA
www.seneca.it
SENECA PACKAGE CD 1.03
INSTALL CoDeSys 2.3
INSTALL Codesys Seneca Library 1.06
INSTALL IsaGRAF
INSTALL Seneca IsaGRAF libraries ver 1.00
INSTALL Java RunTime
INSTALL Seneca Z-NET3 2.00 Beta 12
INSTALL Easy Setup 2.02
INSTALL Seneca OPC Server IO 2.00 Beta 1
INSTALL Seneca OPC Server MB Slave 1.02
INSTALL Seneca Z-NET3 RTU 1.50
INSTALL Seneca Z-NET3 Trend Viewer 2.00 Beta 1
VIEW PDFs
Exit

Figure 12 - The installation page of the Seneca package CD

At the first time, click on “install Codesys n.n” to install the Codesys and follow the simple operations; then click on “install Codesys Seneca Library n.n” to install the libraries for Codesys and follow the simple operations.

HOW TO INCLUDE THE SENECA LIBRARIES INTO CODESYS

When a project is opened, in the CoDeSys toolbar select Window->Library Manager: the standard libraries appear.

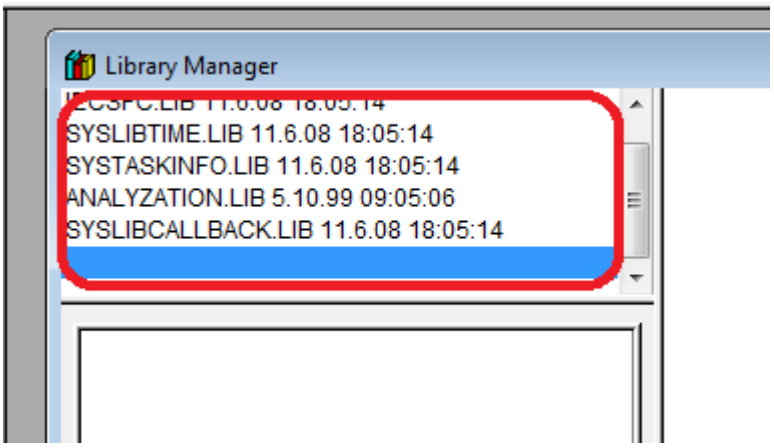


Figure 13 - A particular of the CoDeSys page: list of installed libraries

To insert a Seneca library, right-click into the previous red-signed area and select “Additional library”.

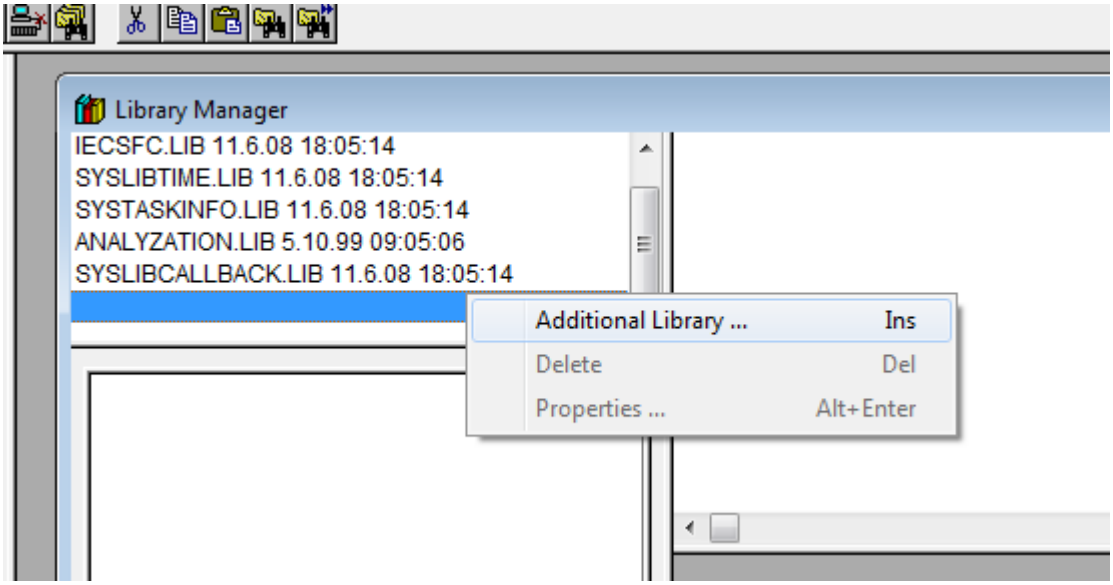


Figure 14 - A particular of the CoDeSys page: how to add libraries

The Seneca libraries are localized in the following link:

C:\...\3S Software\CoDeSys V2.3\Targets\Seneca\Lib_SITEK

So, find the folder “Lib_SITEK” using the combo-box, select the desired library (for example: “TWS_MISC.lib”) and open this one.

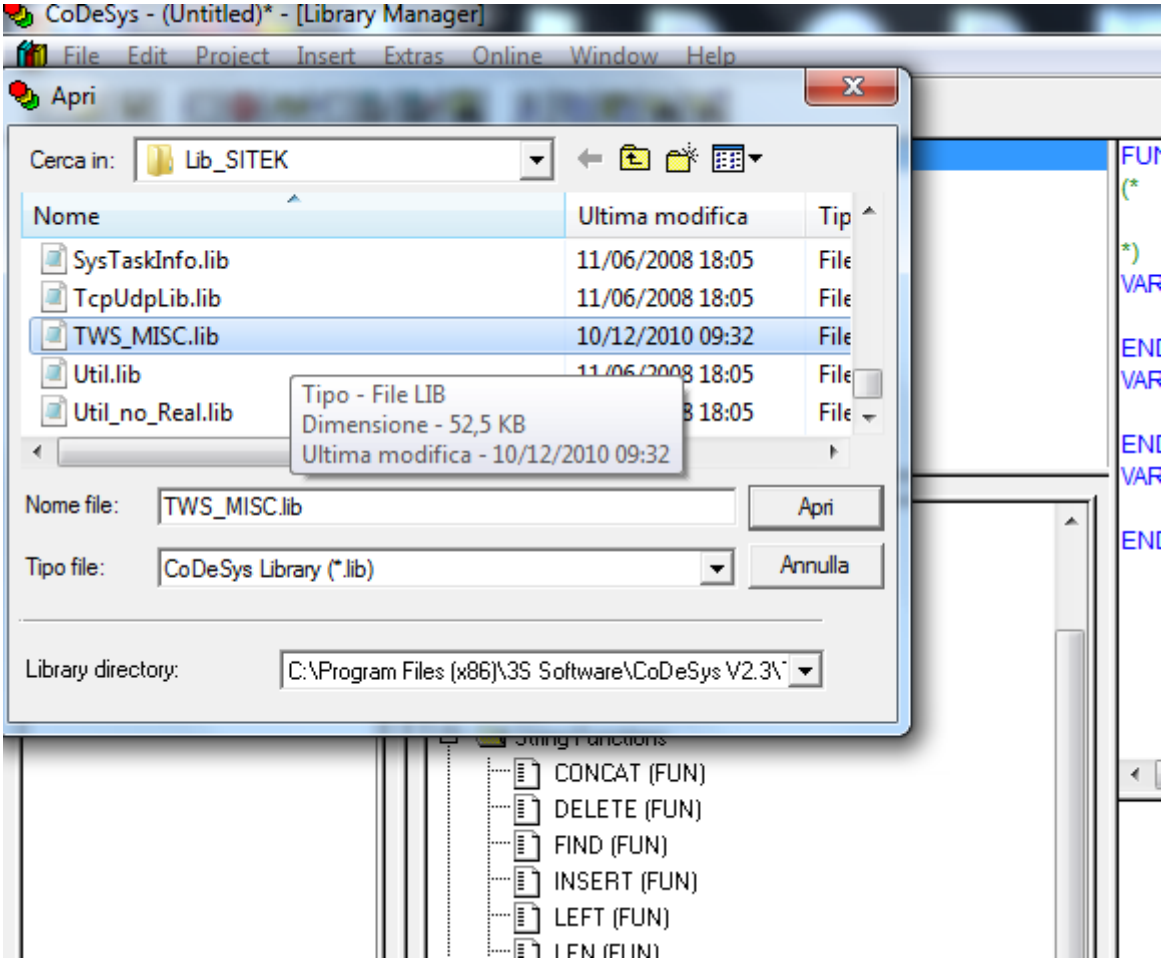


Figure 15 - A particular of the CoDeSys page: TWS_MISC.lib library

An example is shown in the following figure.

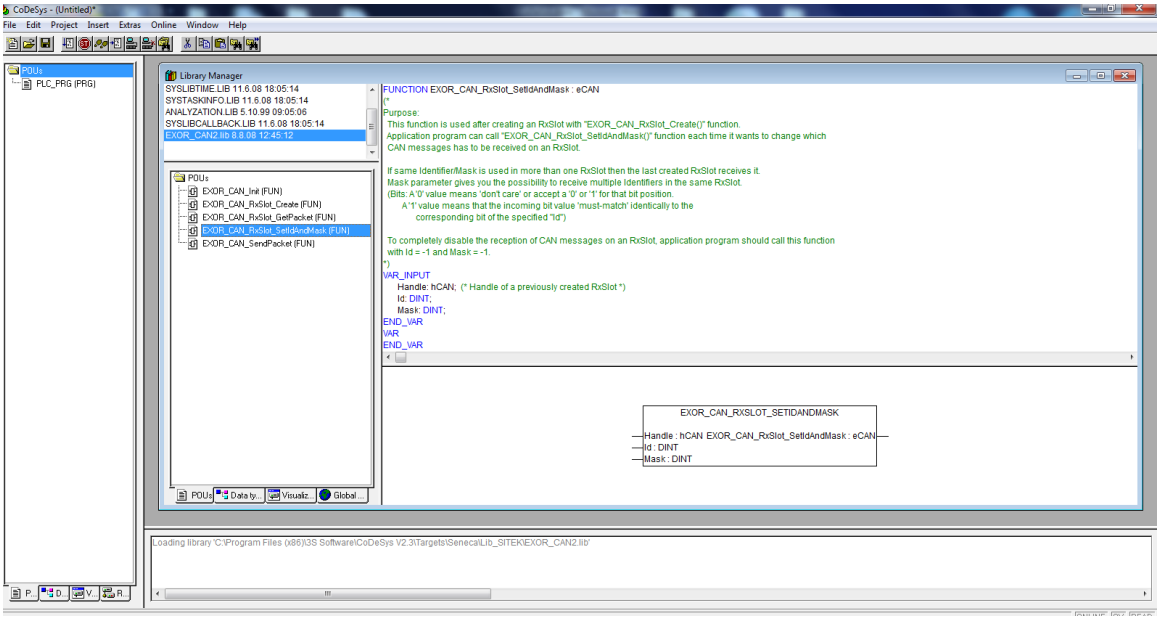

CoDeSys - (Untitled)*
File Edit Project Insert Extras Online Window Help
POUs
PLC_PRG (PRG)
Library Manager
SYSLIBTIME.LIB 11.6.08 18:05:14
SYSTASKINFO.LIB 11.6.08 18:05:14
ANALYZATION.LIB 5.10.99 09:05:06
SYSLIBCALLBACK.LIB 11.6.08 18:05:14
EXOR_CAN2.lib 8.8.08 12:45:12
POUs
EXOR_CAN_Init (FUN)
EXOR_CAN_RxSlot_Create (FUN)
EXOR_CAN_RxSlot_GetPacket (FUN)
EXOR_CAN_RxSlot_SetIdAndMask (FUN)
EXOR_CAN_SendPacket (FUN)
FUNCTION EXOR_CAN_RxSlot_SetIdAndMask : eCAN
(*
Purpose:
This function is used after creating an RxSlot with "EXOR_CAN_RxSlot_Create()" function.
Application program can call "EXOR_CAN_RxSlot_SetIdAndMask()" function each time it wants to change which
CAN messages has to be received on an RxSlot.
If same Identifier/Mask is used in more than one RxSlot then the last created RxSlot receives it.
Mask parameter gives you the possibility to receive multiple Identifiers in the same RxSlot.
(Bits: A '0' value means 'don't care' or accept a '0' or '1' for that bit position.
A '1' value means that the incoming bit value 'must-match' identically to the
corresponding bit of the specified "Id")
To completely disable the reception of CAN messages on an RxSlot, application program should call this function
with Id = -1 and Mask = -1.
*)
VAR_INPUT
Handle: hCAN; (* Handle of a previously created RxSlot *)
Id: DINT;
Mask: DINT;
END_VAR
VAR
END_VAR
EXOR_CAN_RXSLOT_SETIDANDMASK
Handle : hCAN
Id : DINT
Mask : DINT
EXOR_CAN_RxSlot_SetIdAndMask : eCAN
Loading library 'C:\Program Files (x86)\3S Software\CoDeSys V2.3\Targets\Seneca\Lib_SITEK\EXOR_CAN2.lib'

Figure 16 - The CoDeSys page: a detailed visualization of the libraries

In the appendix B are described the Seneca libraries that can be used to CoDeSys.

Configuring Z-TWS with I/O Modbus modules

The Z-TWS permits to use CoDeSYS like a PLC developing environment; but first of all, it is necessary configure Z-TWS Serial Ports and to assign address e baudrate to I/O modules. All these steps are possible using the softwares ZNET (as configurator) and Codesys, as shown in the following block diagram.

Figure 17 – Block diagram process to configure the Z-TWS with I/O Modbus

For example, open Seneca ZNET and click on new project, the following window appears.

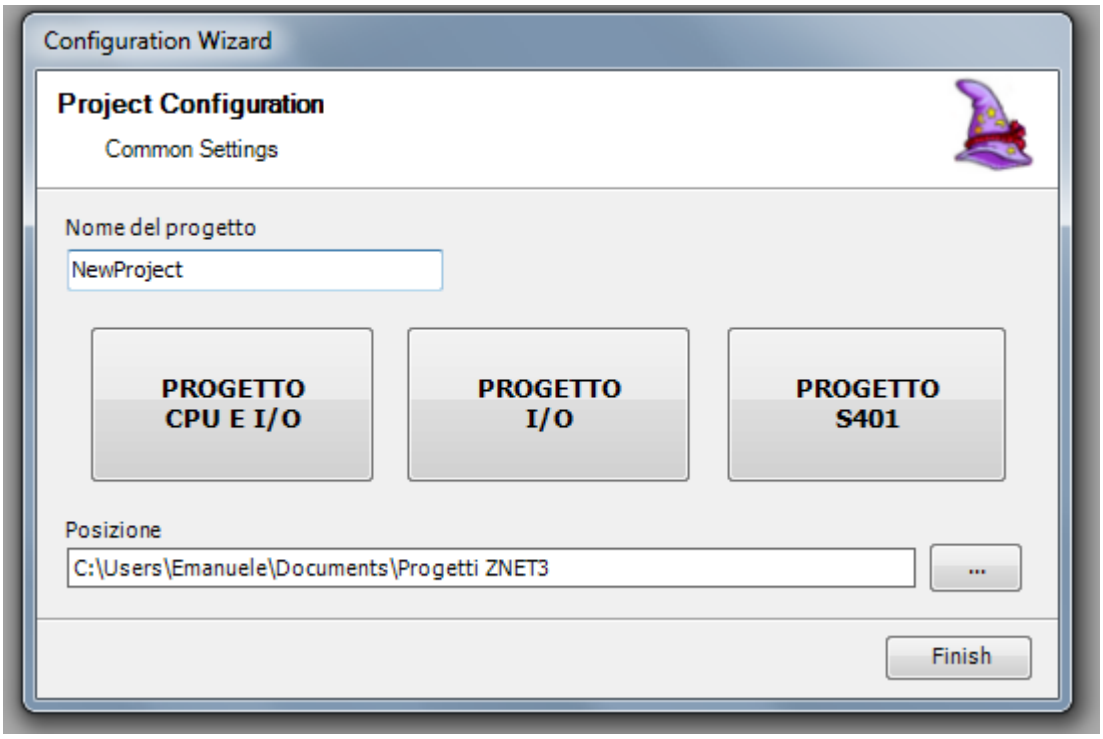


Figure 18 - Configuring the Z-TWS with I/O Modbus – step 1

Select “Project CPU and I/O”, and then “Project CoDeSys”; for example, now, select Z-TWS3 as CPU unit. Moreover, choose the “target” depending on the firmware version of libraries.

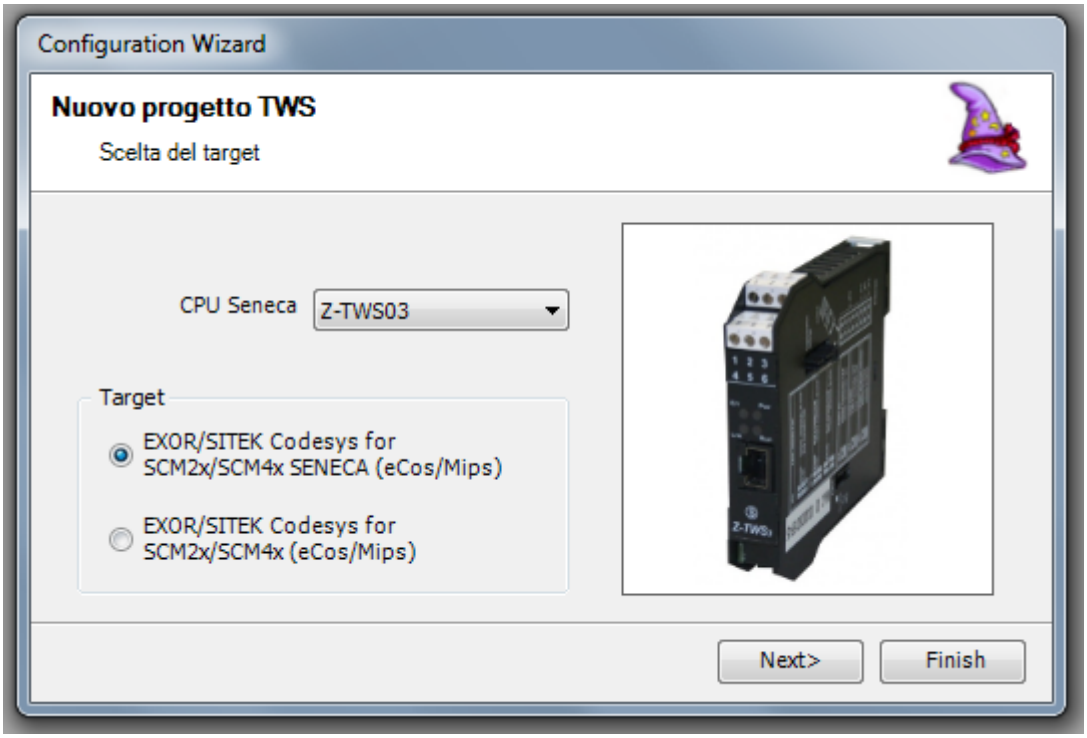


Figure 19 - Configuring the Z-TWS with I/O Modbus – step 2

Click “next” and the following window appears:

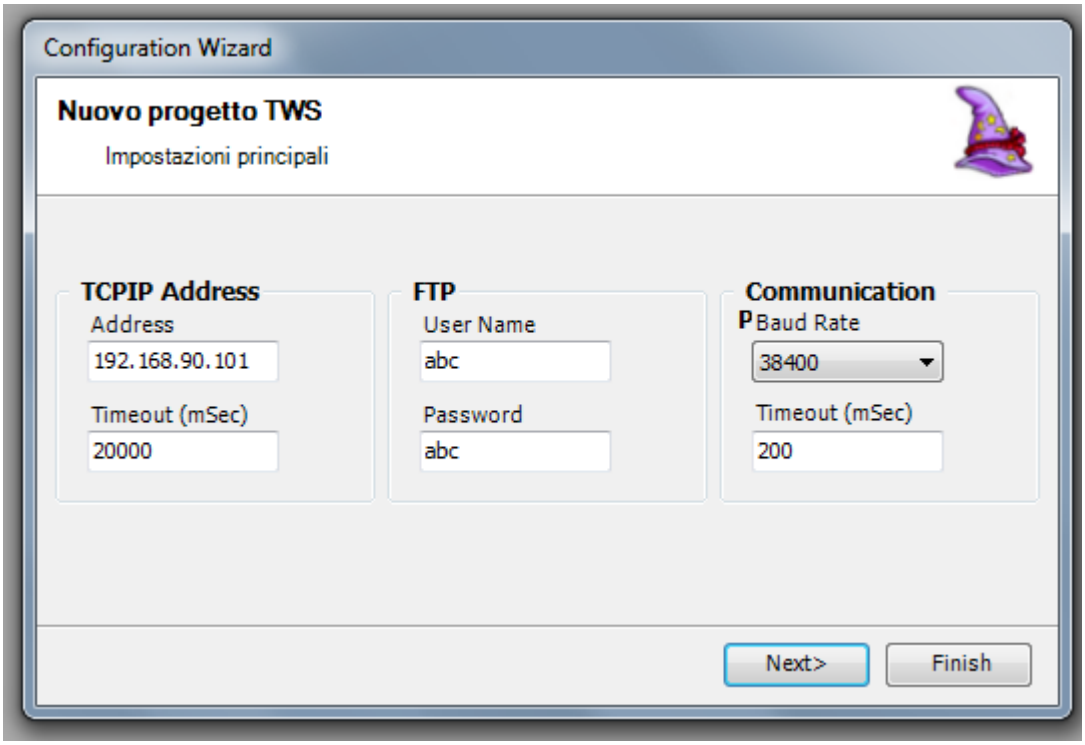


Figure 20 - Configuring the Z-TWS with I/O Modbus – step 3

When the project is opened, it add automatically CPU with IP ADDRESS 192.168.90.101; at this point click “Next”

For example select “Inserted off-line” in the following window:

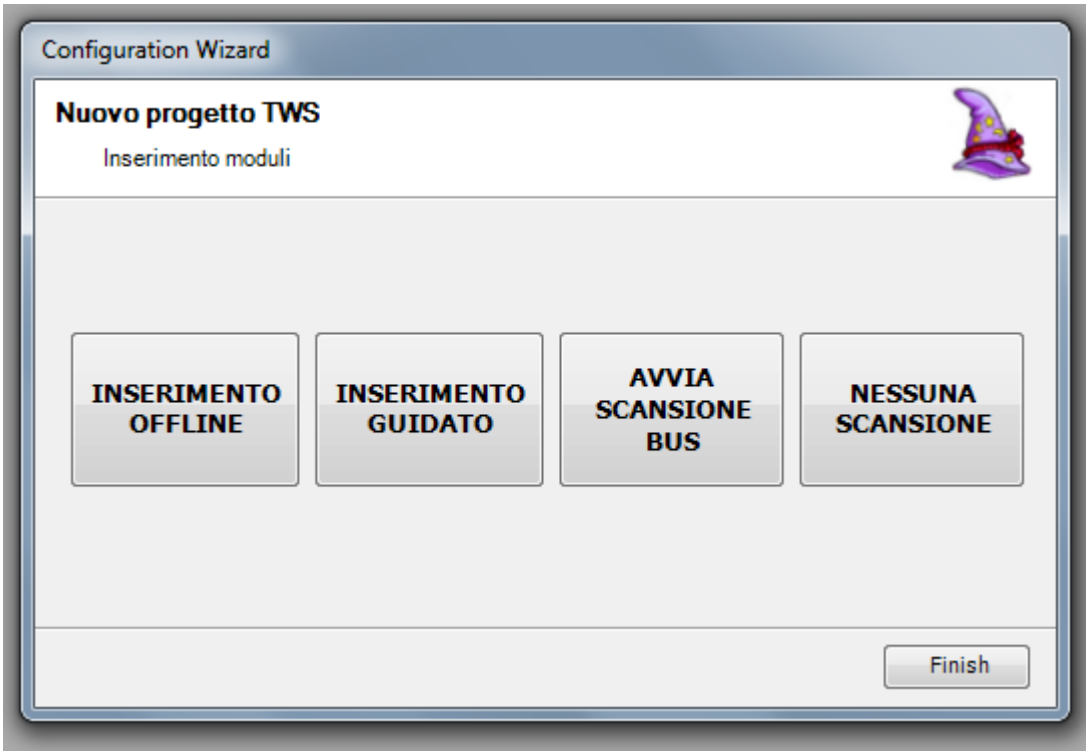


Figure 21 - Configuring the Z-TWS with I/O Modbus – step 4

Now, the “type of module” window allows to insert all the desired Seneca Z-PC Line modules in the new project opened.

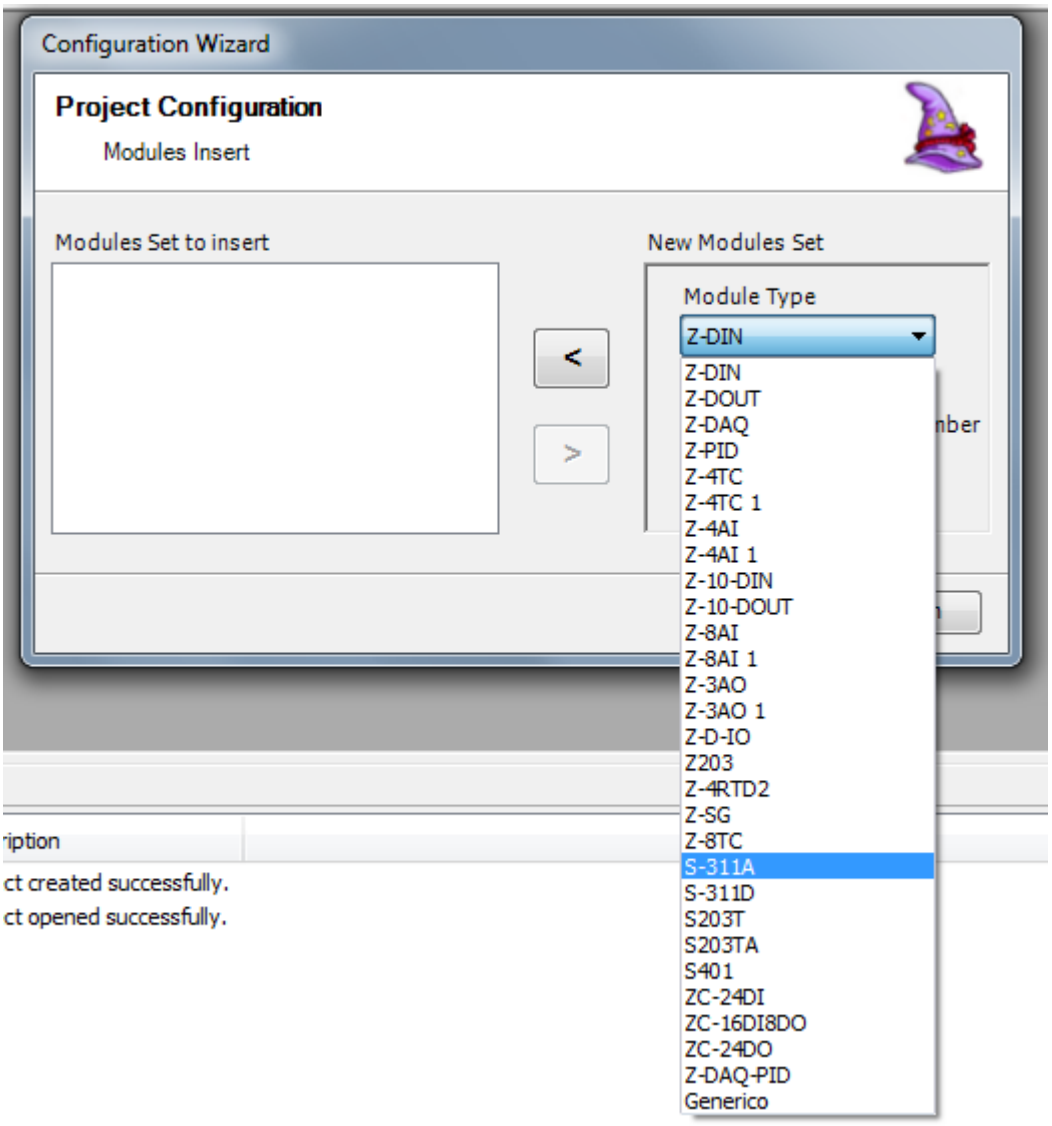


Figure 22 - Configuring the Z-TWS with I/O Modbus – step 5

Click on the “end” button and the following configuration page appears (only an example):

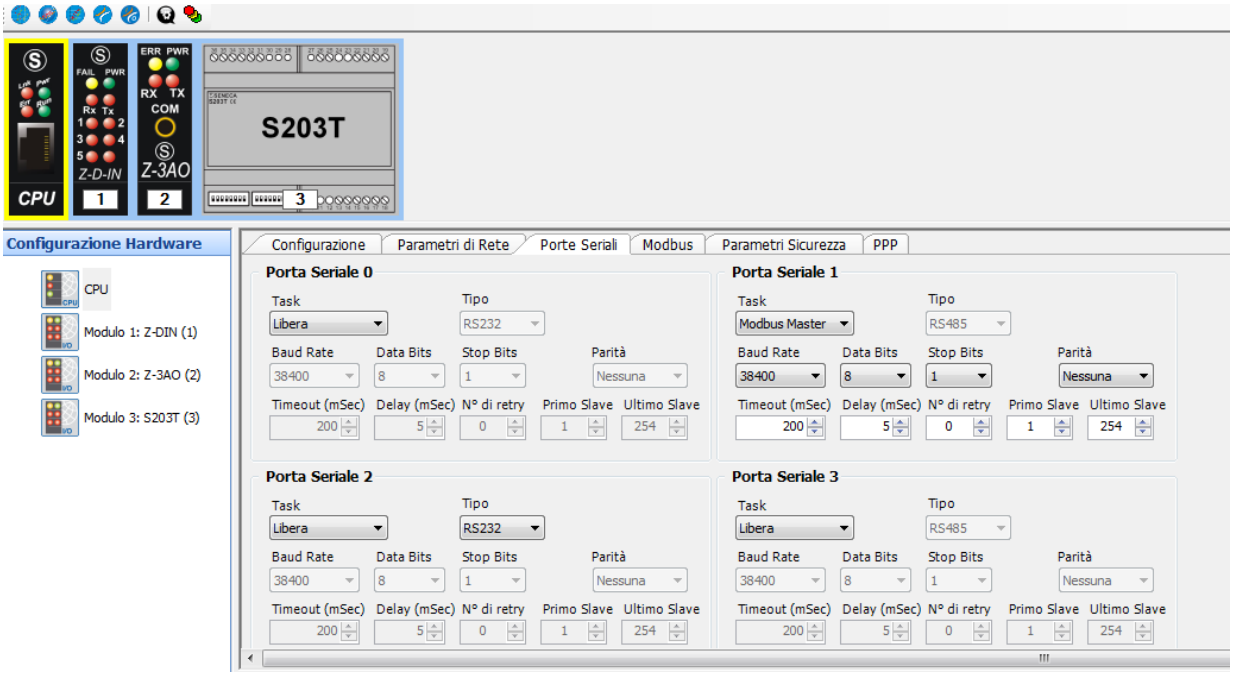


Figure 23 - Configuring the Z-TWS with I/O Modbus – step 6

In particular, it is important to set the port #1 (RS485 bus), because it allows communication with I/O modules by IDC10 rear bus connector.

IMPORTANT: **The serial port configurations must correspond to the jumper setting of Z-TWS (see the chapter "jumper setting").**

Tipically (default) the serial port 2 is configurated as "RS232" and it is used to communicate with the modem (this port is related to the "PPP enabled" flag).

IMPORTANT!

How To Evaluate the Modbus Distributed System Performances

The data communication between Master and Slaves using the ModBUS-RTU protocol is based on queries/replies between one Master and each Slave, as shown in the following example.

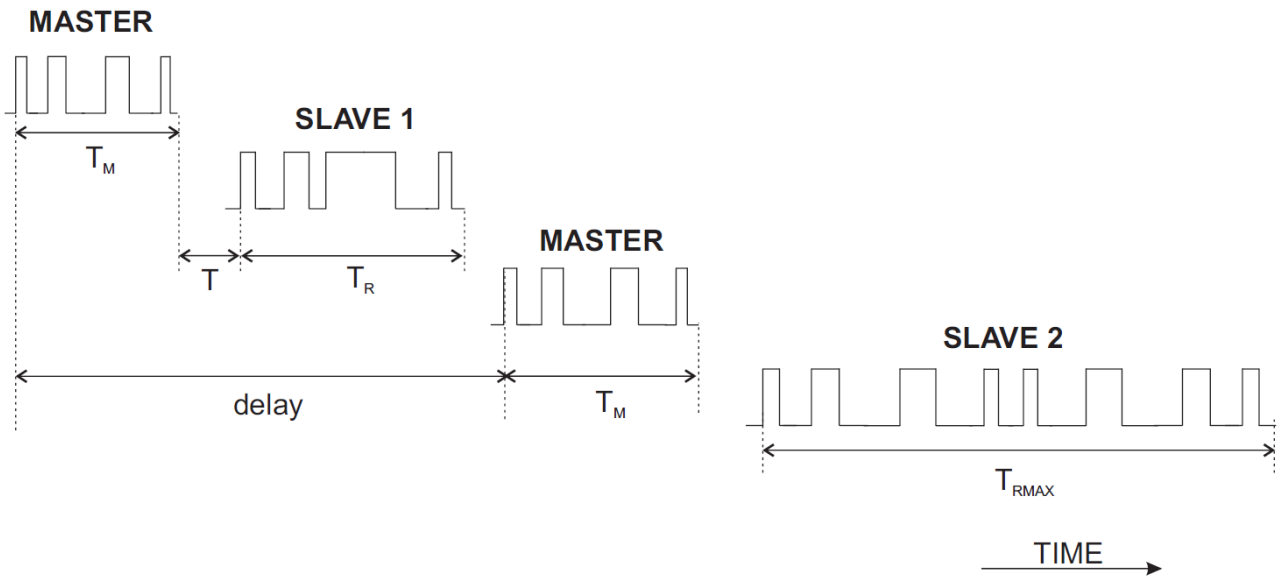


Figure 24 - An example of Master and slaves communication (Modbus)

- **In particular, the time between the end of a Master query and the start of a Slave response (T) needs to be greater than "Timeout" parameter in ZNET.**
- **The processing time of the Master (T_M) results a constant value approximately , while the Slave response time (T_{RMAX}) is a function of the Baud Rate and number of registers queried. However, this value may be estimated as 1/BaudRate.**

So, the following formula permits to estimate the minimum value of the delay (see figure 23; parameter in ZNET, with default value 10 ms):

$$delay > delay_{MIN} = T + T_{RMAX} + T_M + T_{PLC}$$

where T_{PLC} is the time employed to perform the phases of PLC (reading – execution - writing).

$$T_{PLC} = T_{TOTREADING} + T_{EXE} + T_{TOTWRITING}$$

where $T_{TOTREADING}$ means the total time to perform the reading phase, and it is

$$T_{TOTREADING} = \text{delay X number of slave modules}$$

where T_{EXE} means the total time to perform the execution phase and $T_{TOTWRITING}$ means the total time to perform the writing phase, and it is

$$T_{TOTWRITING} = T_{WRITING} \text{ X number of slave modules}$$

and $T_{WRITING} = T_{MASTER} + T + T_{SLAVE}$ (during writing).

After this, select the “Network parameters” window: the default value of IP address is 192.168.90.101.

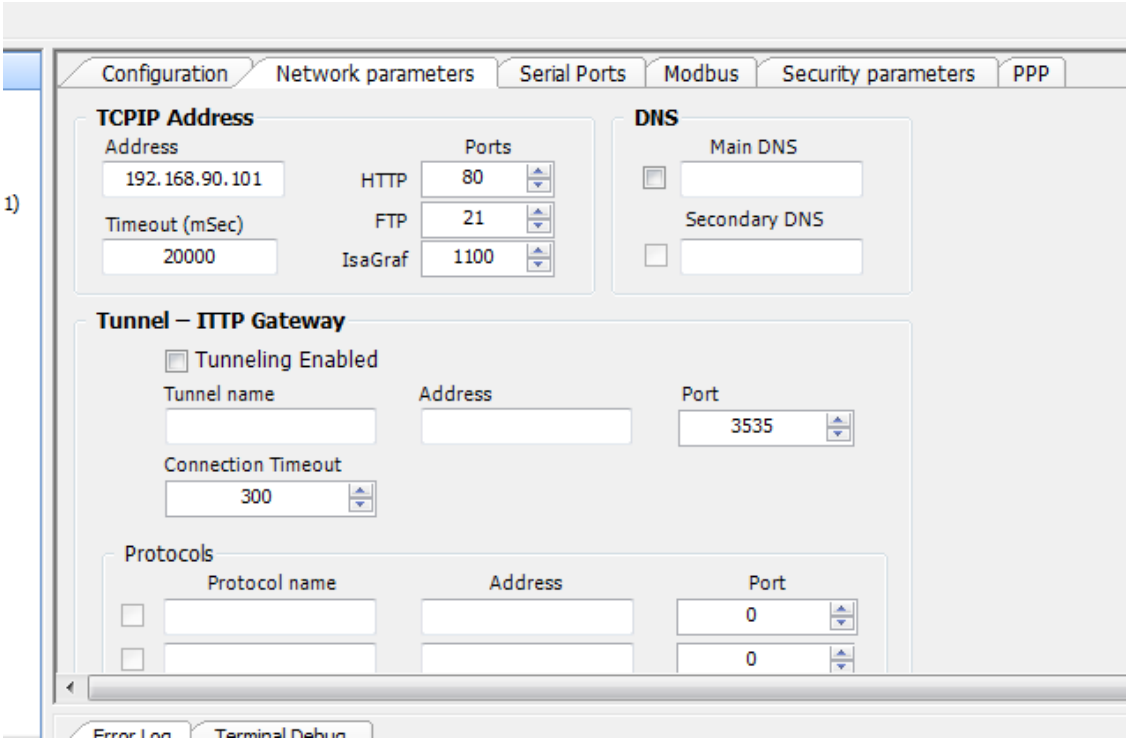


Figure 25 - Configuring the Z-TWS with I/O Modbus – step 7

After you have saved settings, the Z-NET software produces a configuration file for the Data Acquisition Module, called “PLC.ini”: click on Download CPu Configuration and click on the following button to export the configuration in Codesys.

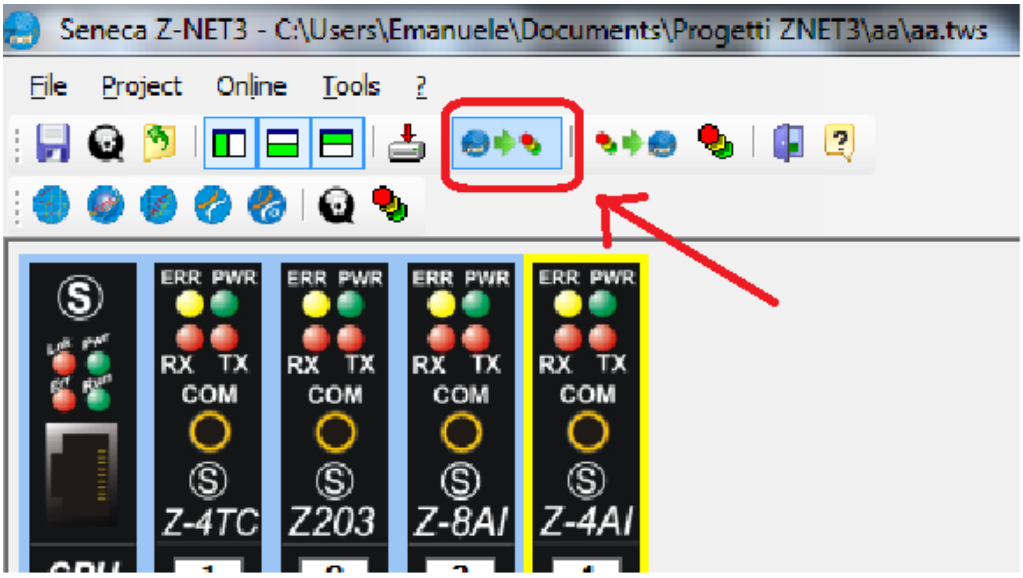


Figure 26 - Configuring the Z-TWS with I/O Modbus – step 8

At this point, Codesys is opened; from menu, select Online->Communication parameters and the following window will be opened. Click on “New” to add a new channel and select “TCP/IP”.

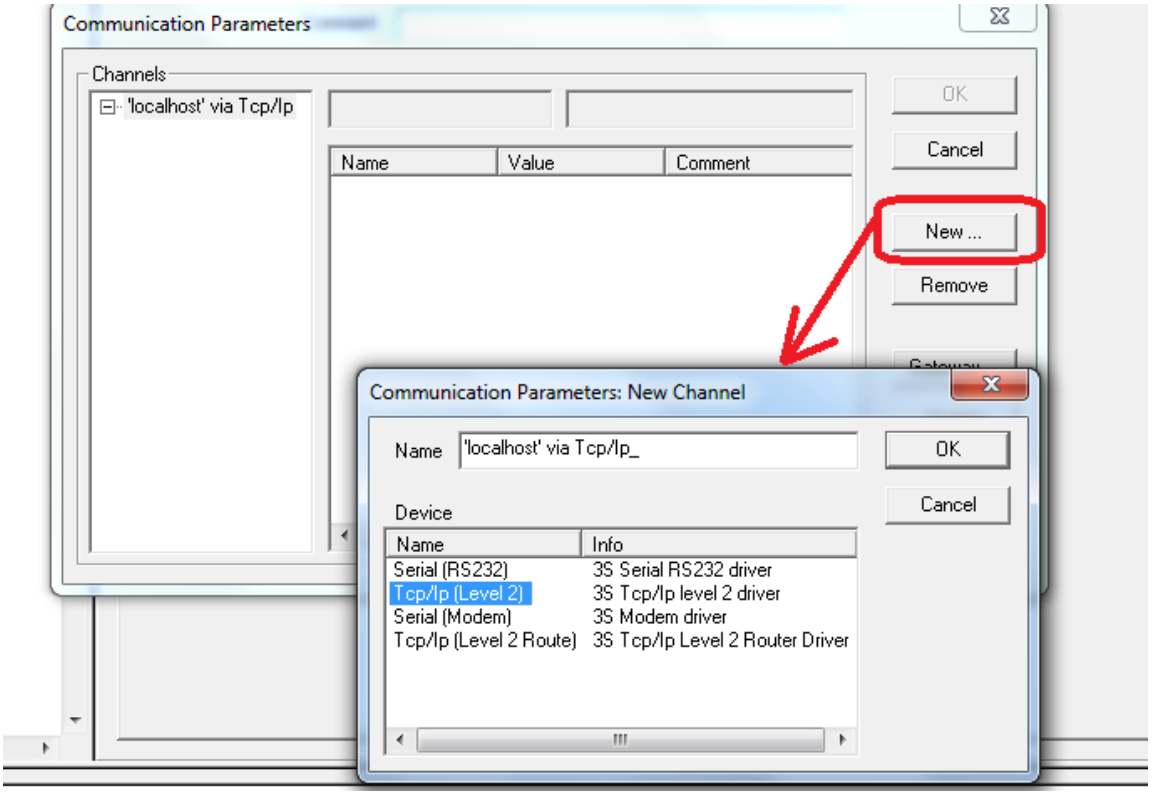


Figure 27 - Configuring the Z-TWS with I/O Modbus – step 9

At this point, replace the “localhost” string with the IP of the TWS used, for example 192.168.90.101.

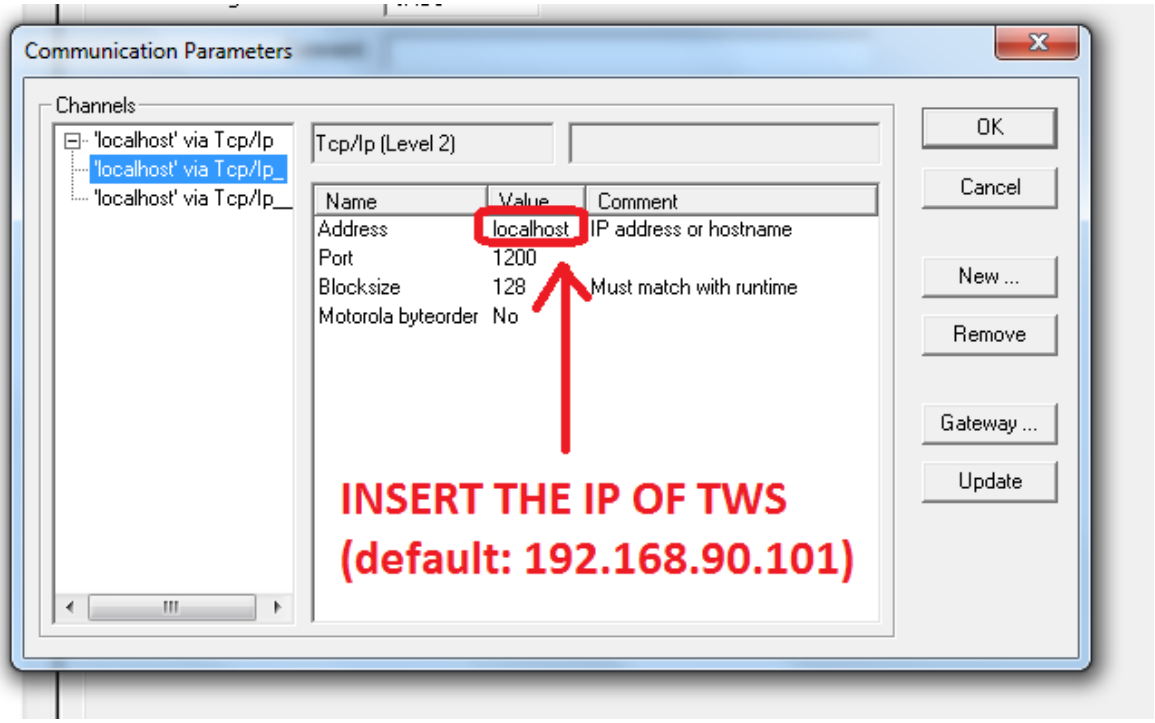


Figure 28 - Configuring the Z-TWS with I/O Modbus – step 10

With reference to the previous window click OK; then, select the “Resources” window in the left-side of the main page

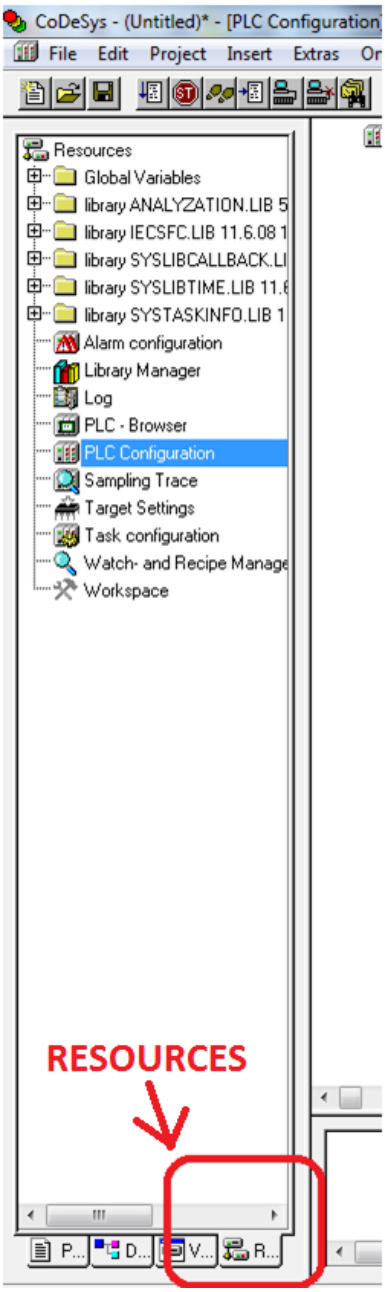


Figure 29 - Configuring the Z-TWS with I/O Modbus – step 11

Select “PLC Configuration” and the following window appears: as you can see, there is a list of modules opened. For each module, the exported variables can be viewed.

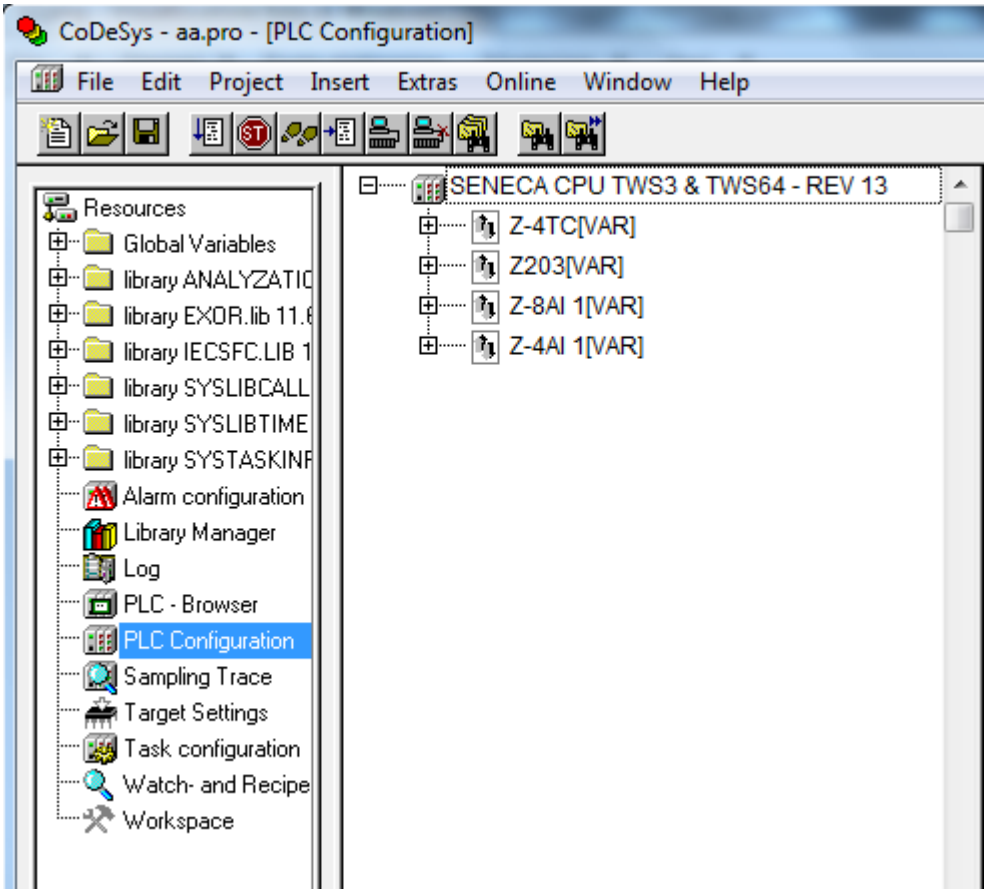


Figure 30 - Configuring the Z-TWS with I/O Modbus – step 11

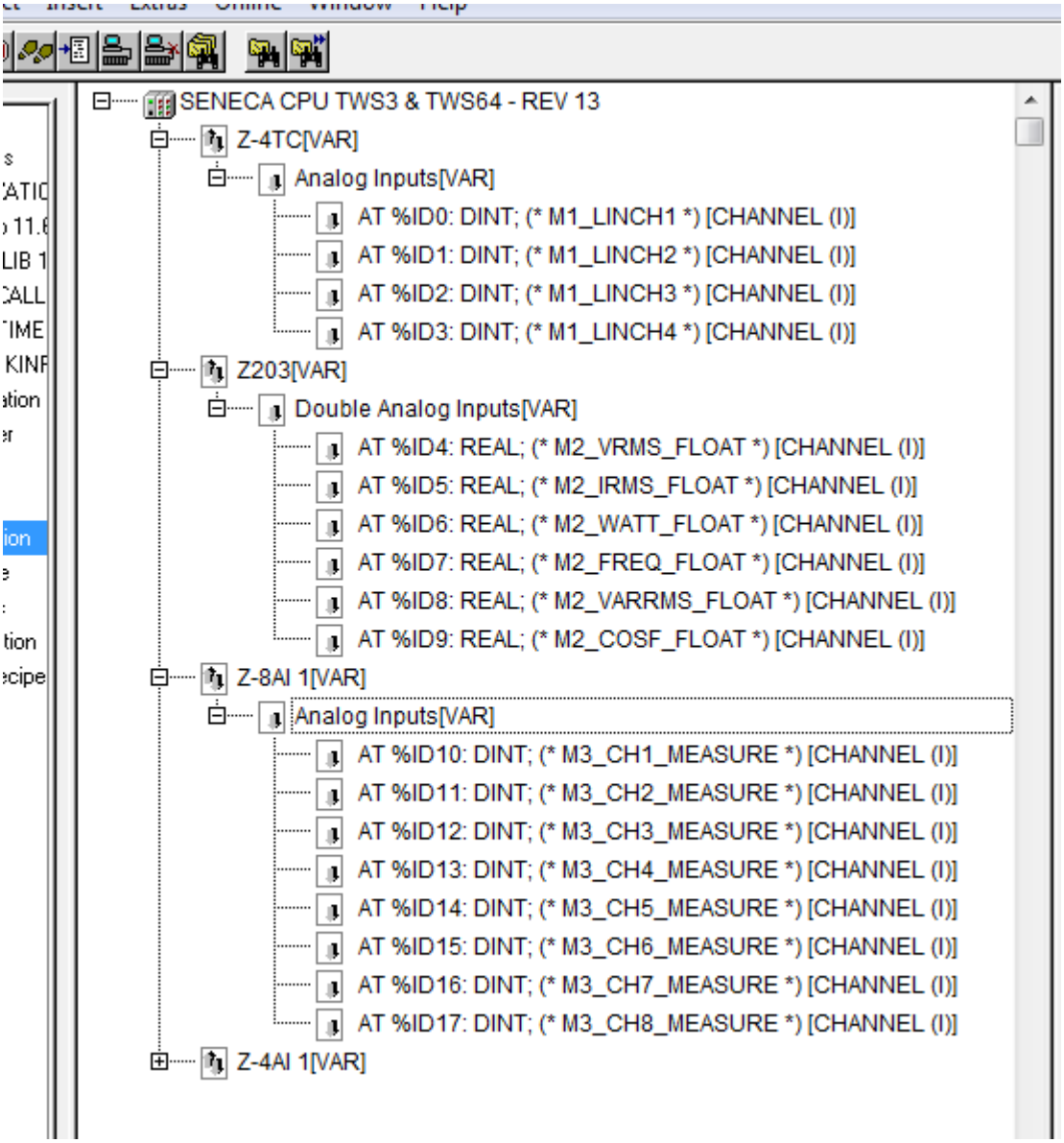

SENECA CPU TWS3 & TWS64 - REV 13
Z-4TC[VAR]
Analog Inputs[VAR]
AT %ID0: DINT; (* M1_LINCH1 *) [CHANNEL (I)]
AT %ID1: DINT; (* M1_LINCH2 *) [CHANNEL (I)]
AT %ID2: DINT; (* M1_LINCH3 *) [CHANNEL (I)]
AT %ID3: DINT; (* M1_LINCH4 *) [CHANNEL (I)]
Z203[VAR]
Double Analog Inputs[VAR]
AT %ID4: REAL; (* M2_VRMS_FLOAT *) [CHANNEL (I)]
AT %ID5: REAL; (* M2_IRMS_FLOAT *) [CHANNEL (I)]
AT %ID6: REAL; (* M2_WATT_FLOAT *) [CHANNEL (I)]
AT %ID7: REAL; (* M2_FREQ_FLOAT *) [CHANNEL (I)]
AT %ID8: REAL; (* M2_VARRMS_FLOAT *) [CHANNEL (I)]
AT %ID9: REAL; (* M2_COSF_FLOAT *) [CHANNEL (I)]
Z-8AI 1[VAR]
Analog Inputs[VAR]
AT %ID10: DINT; (* M3_CH1_MEASURE *) [CHANNEL (I)]
AT %ID11: DINT; (* M3_CH2_MEASURE *) [CHANNEL (I)]
AT %ID12: DINT; (* M3_CH3_MEASURE *) [CHANNEL (I)]
AT %ID13: DINT; (* M3_CH4_MEASURE *) [CHANNEL (I)]
AT %ID14: DINT; (* M3_CH5_MEASURE *) [CHANNEL (I)]
AT %ID15: DINT; (* M3_CH6_MEASURE *) [CHANNEL (I)]
AT %ID16: DINT; (* M3_CH7_MEASURE *) [CHANNEL (I)]
AT %ID17: DINT; (* M3_CH8_MEASURE *) [CHANNEL (I)]
Z-4AI 1[VAR]

Figure 31 - Configuring the Z-TWS with I/O Modbus – step 12

To download the program, select the Online->Login from menu.

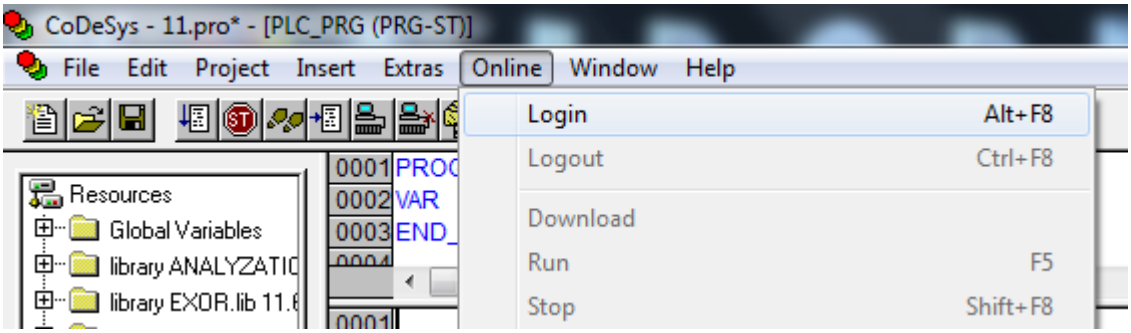


Figure 32 - Configuring the Z-TWS with I/O Modbus – step 13

To run the PLC program, it is necessary to select Online->Run.

Configuring the Z-TWS with I/O CANopen

Codesys allows to configure a TWS project based on the CANopen bus using the Seneca I/O modules (for example: ZC-8AI, ZC-SG, etc…) that supports CANopen protocol; instead for more information about the use of Isagraf for CANopen, see the appendix C.

Figure 33 - Block diagram process to configure the Z-TWS with I/O CANopen

For example, launch Codesys, and create a new project (press file/NEW), the following window appears (select one of the two following options)

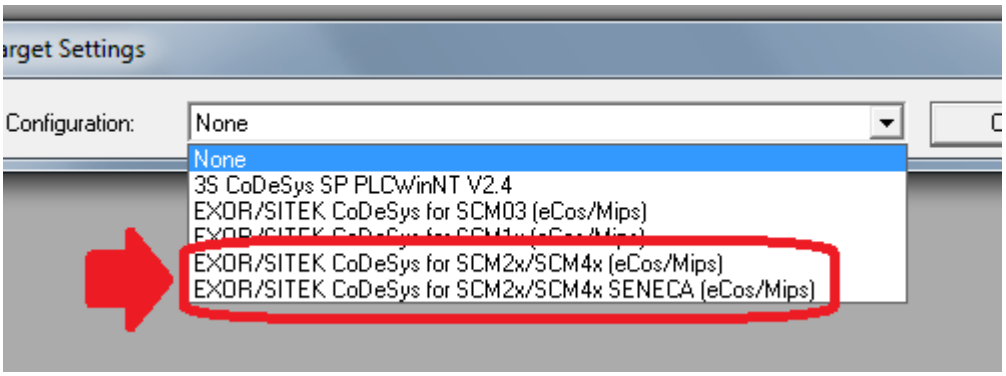


Figure 34 - Configuring the Z-TWS with I/O CANopen – step 1

The features of the target settings for two (alternative) previous cases are shown in the following figures:

SCM4x (eCos/Mips) –

if you have selected the first target configuration

NOTE: USE THIS CONFIGURATION IF FIRMWARE VERSION IS 3.XX

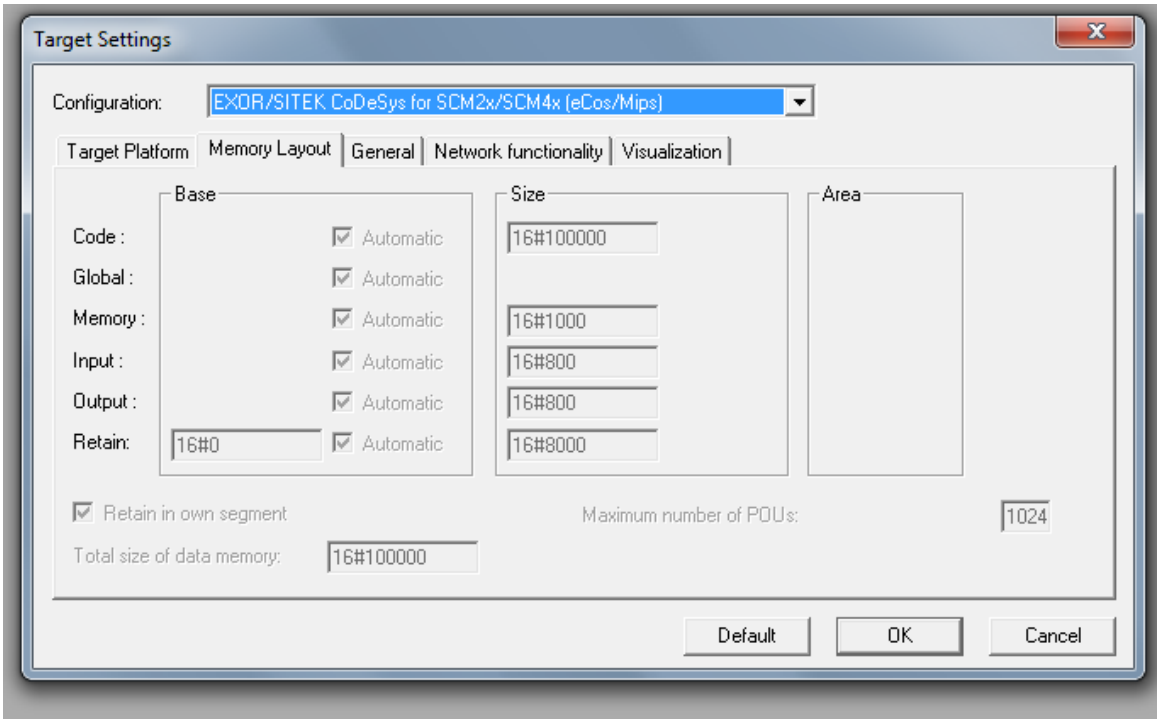


Figure 35 - Configuring the Z-TWS with I/O CANopen – step 2A

The following table shows the Memory layout for this configuration

	Size (hex) bytes	Size (decimal) bytes
Code	100000	1048576
Global		
Memory	1000	4096
Input	800	2048
Output	800	2048
Retain	8000	32768

SCM4x SENECA (eCos/Mips) –

if you have selected the second target configuration

NOTE: USE THIS CONFIGURATION IF FIRMWARE VERSION IS 4.XX

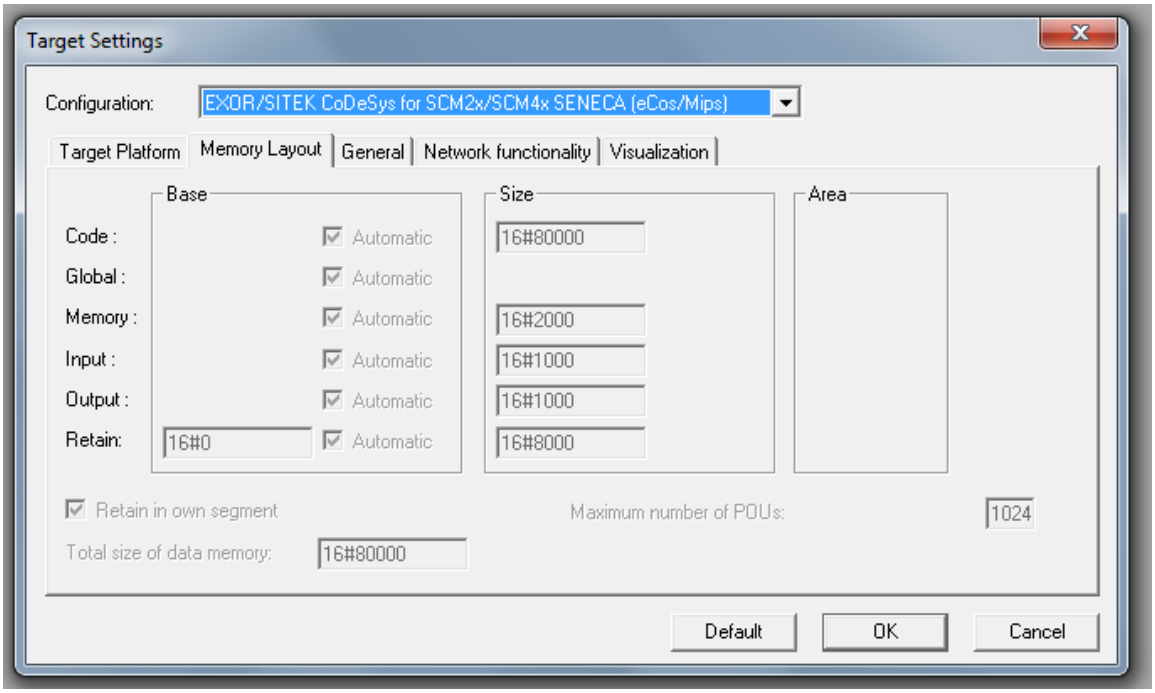


Figure 36 - Configuring the Z-TWS with I/O CANopen – step 2B

The following table shows the Memory layout for this configuration

	Size (hex) bytes	Size (decimal) bytes
Code	80000	524288
Global		
Memory	2000	8192
Input	1000	4096
Output	1000	4096
Retain	8000	32768

At this point, configure a new POU as follows (Name, Type, Language) and click OK

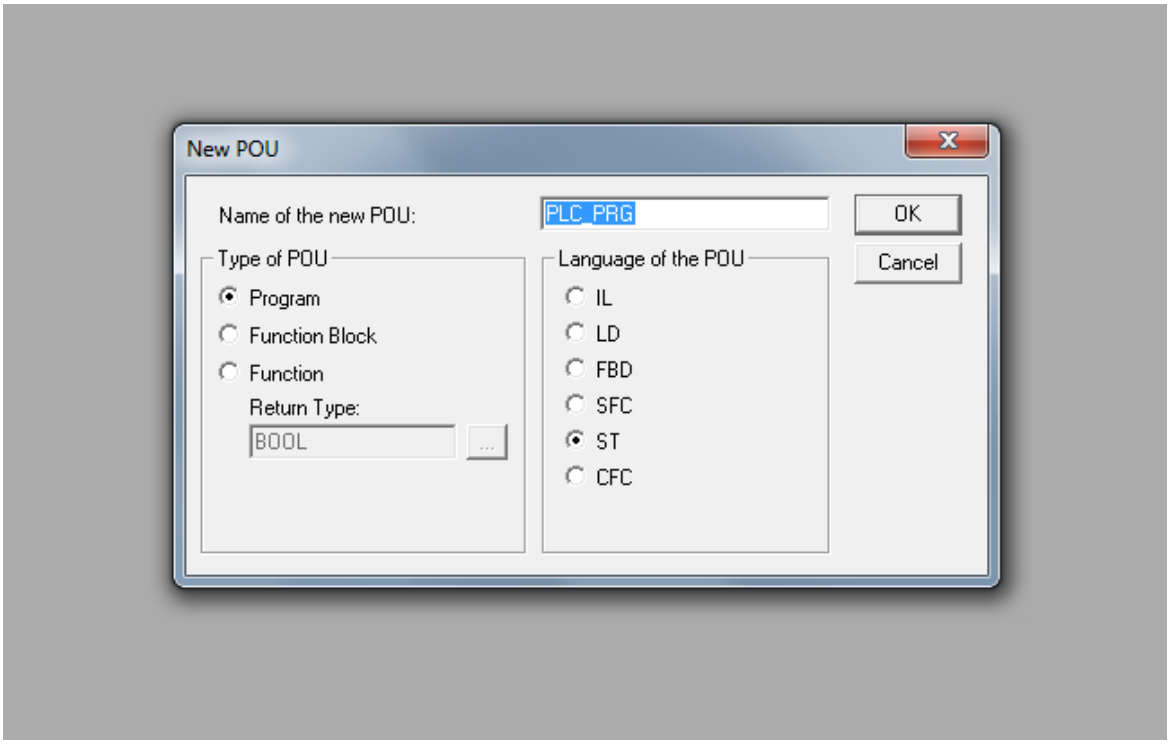


Figure 37 - Configuring the Z-TWS with I/O CANopen – step 3

The previous window permits to initialize POu (Program Organization units: Program, functions block and functions): in particular the PLC_PRG, is generally the main program in a single task program. Each project must contain this POu, which is called exactly once per control cycle.

At this point, a new project of Codesys is opened; from menu, select Online->Communication parameters and the following window will be opened. Click on “New” to add a new channel and select “TCP/IP”.

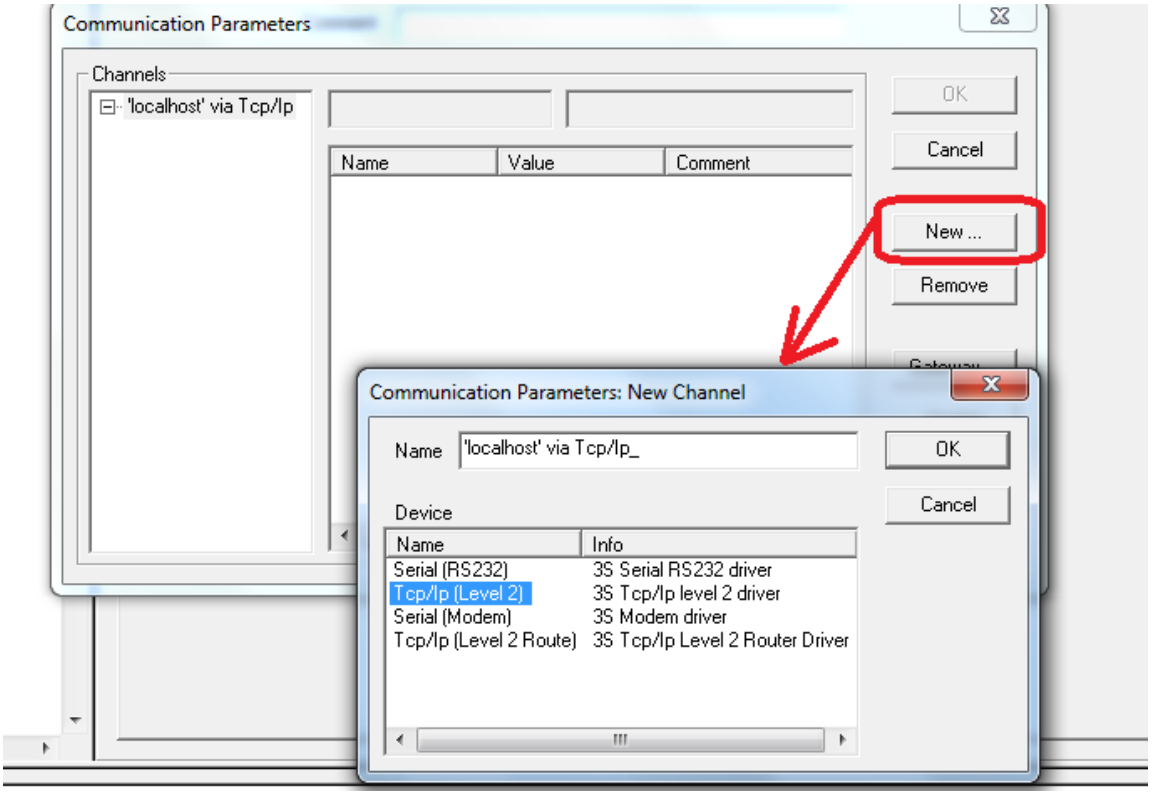


Figure 38 - Configuring the Z-TWS with I/O CANopen – step 4

At this point, replace the “localhost” string with the IP of the TWS used, for example 192.168.90.101.

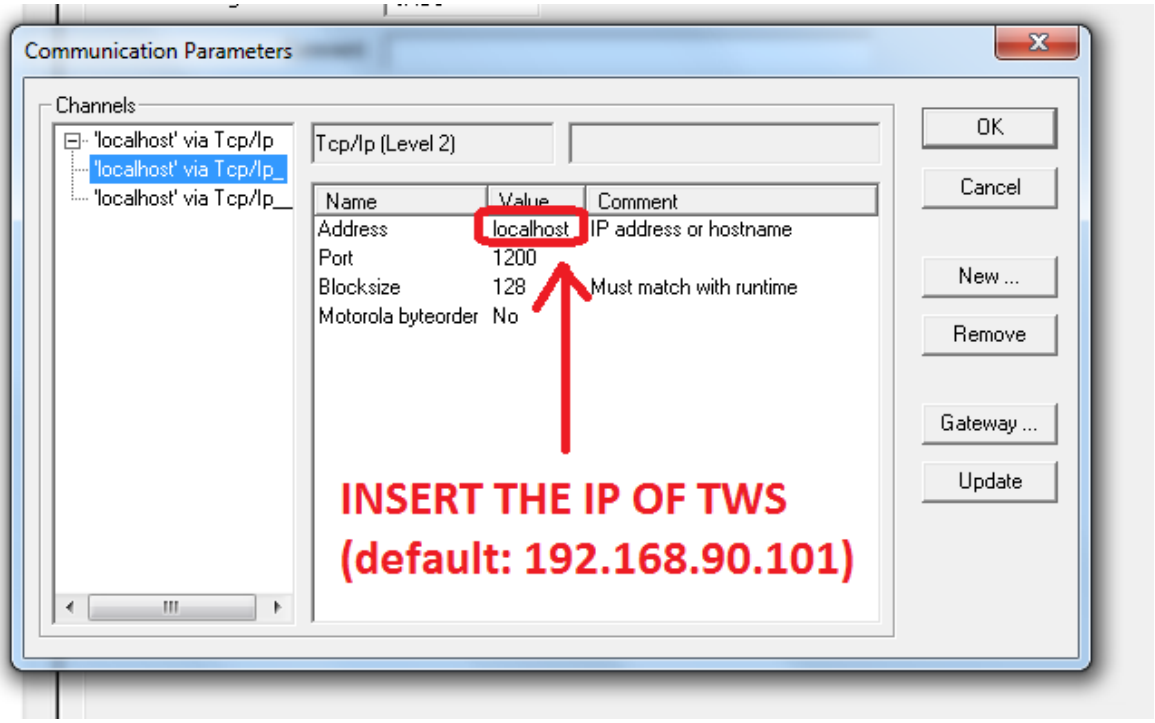


Figure 39 - Configuring the Z-TWS with I/O CANopen – step 5

With reference to the previous window click OK; at this point, in the following page, write the program into this page (because at least one statement is required, otherwise an error occurs); for example, digit ";" in the line 1 and save.

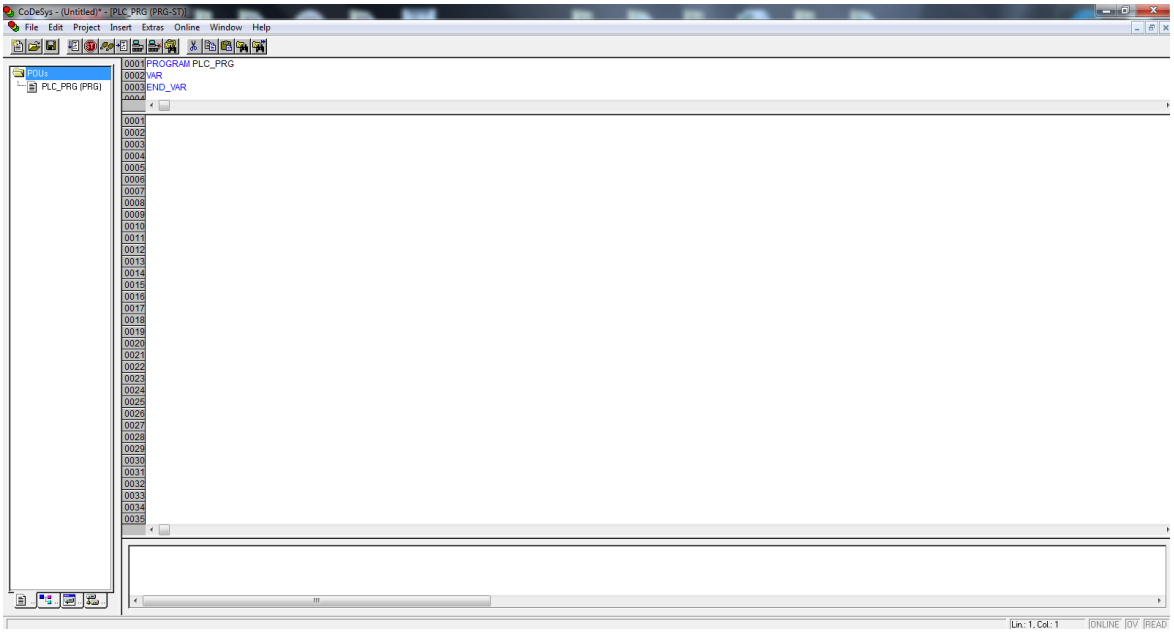


Figure 40 - Configuring the Z-TWS with I/O CANopen – step 6

Select the window “Resources” in the left-side of the previous page and appears the following list

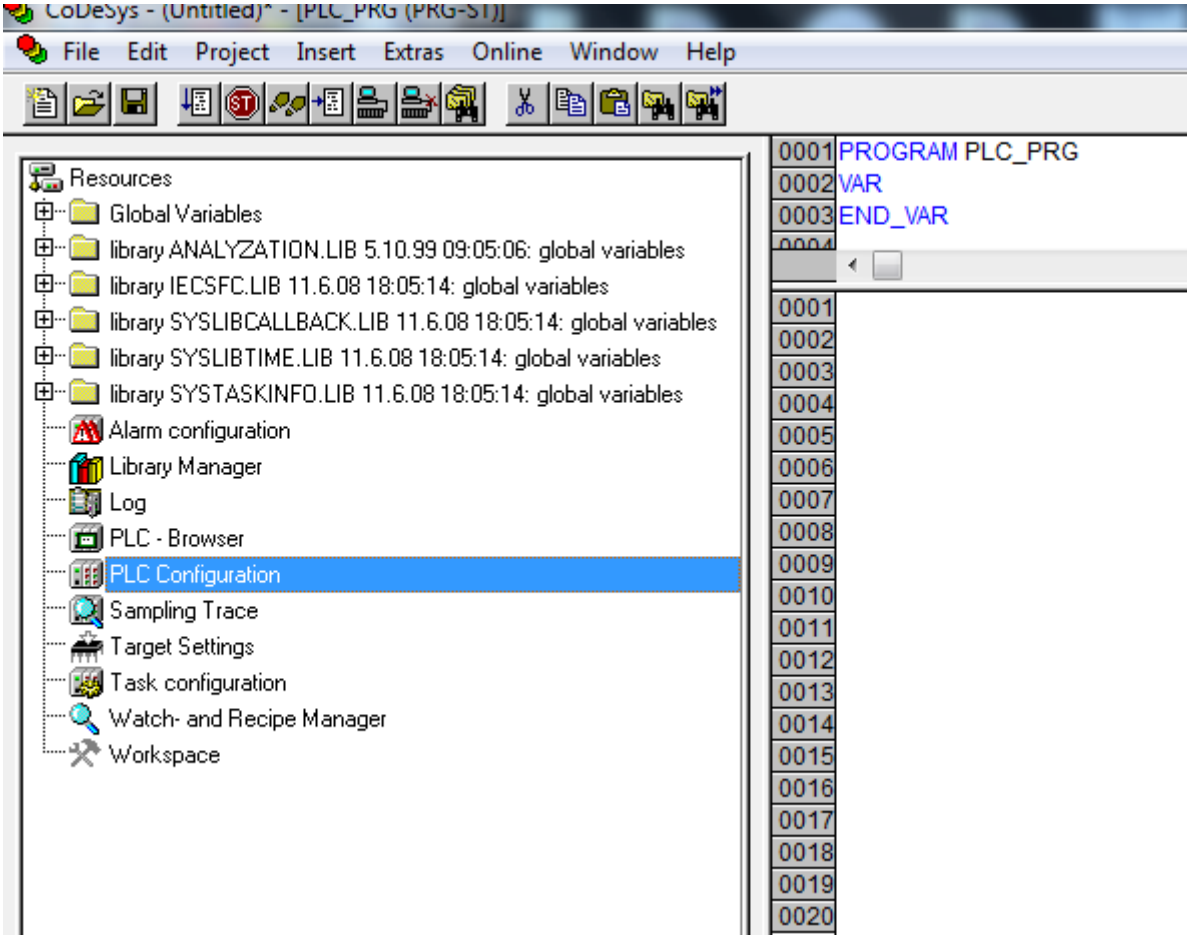


Figure 41 - Configuring the Z-TWS with I/O CANopen – step 7

Double click the “PLC Configuration” voice and appears the following

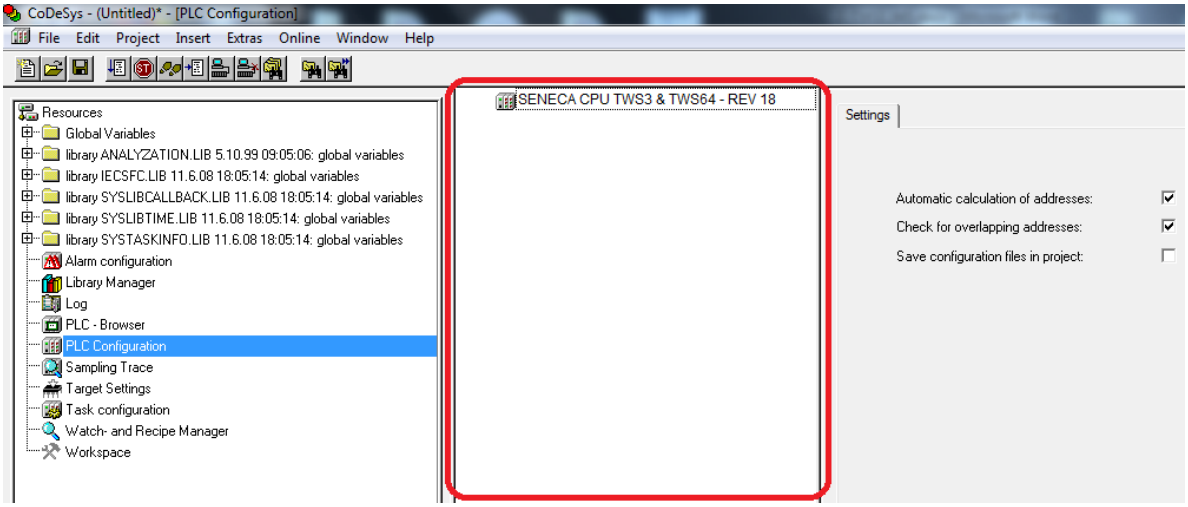


Figure 42 - Configuring the Z-TWS with I/O CANopen – step 8

To insert the CAN master, right-click into the previous red-signed area and select “Append subelement”.

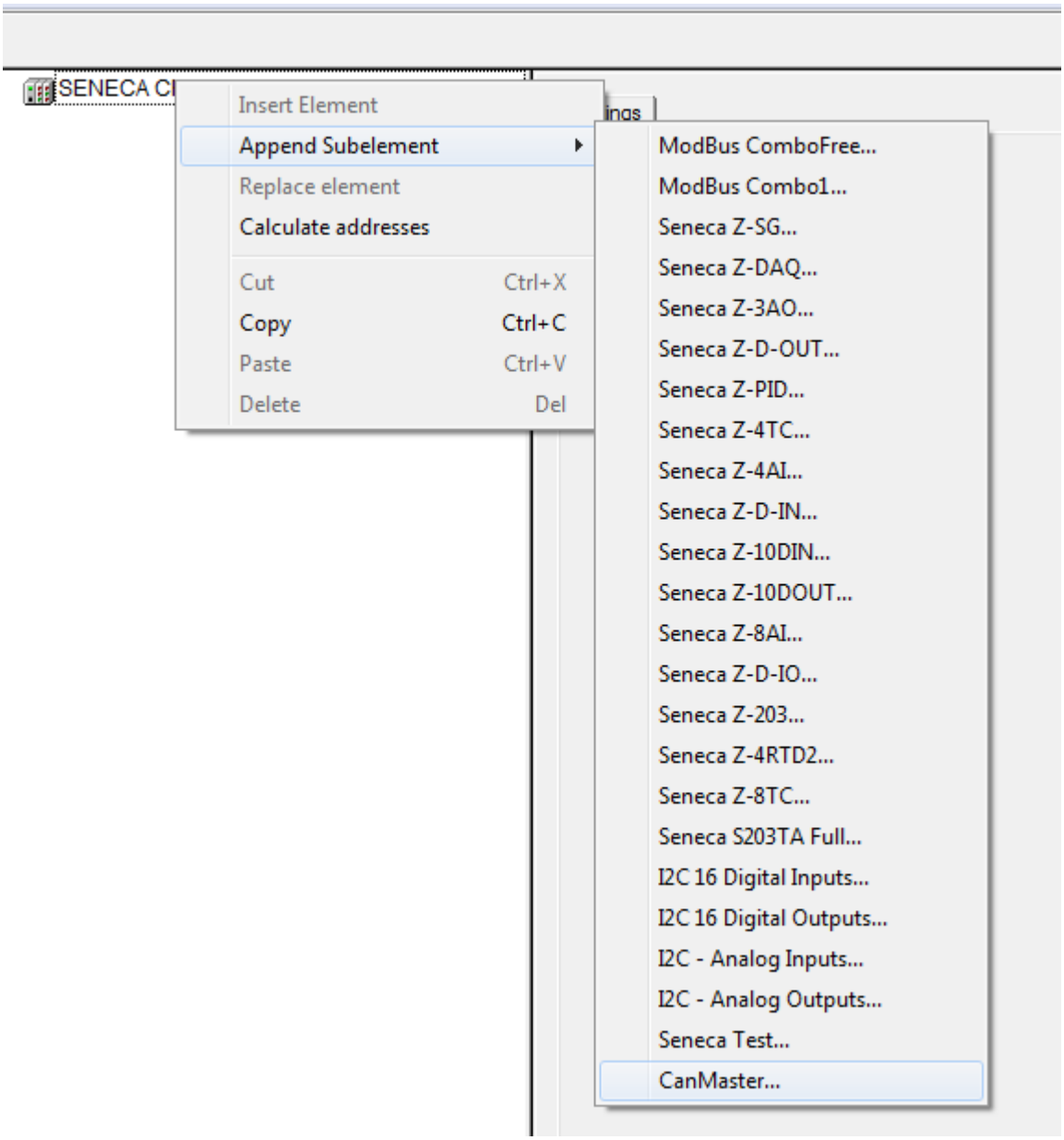


Figure 43 - Configuring the Z-TWS with I/O CANopen – step 9

Configure the CAN parameters, as the baudrate

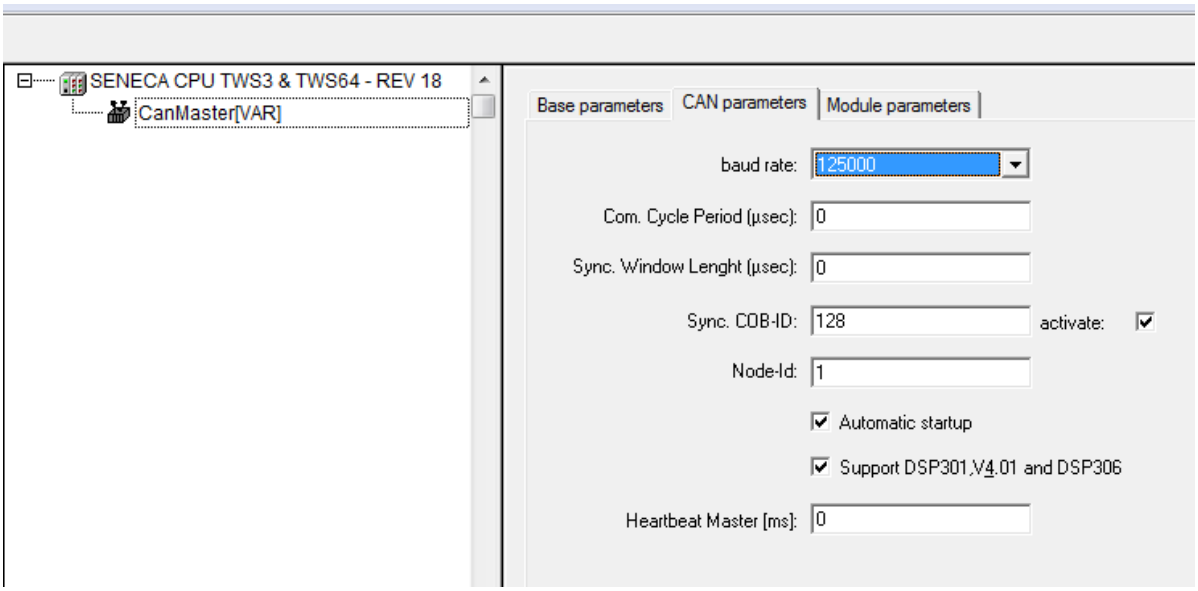


Figure 44 - Configuring the Z-TWS with I/O CANopen – step 10

To insert the I/O Seneca module based on CANopen, right-click into the “CanMaster” line and select the desired modules.

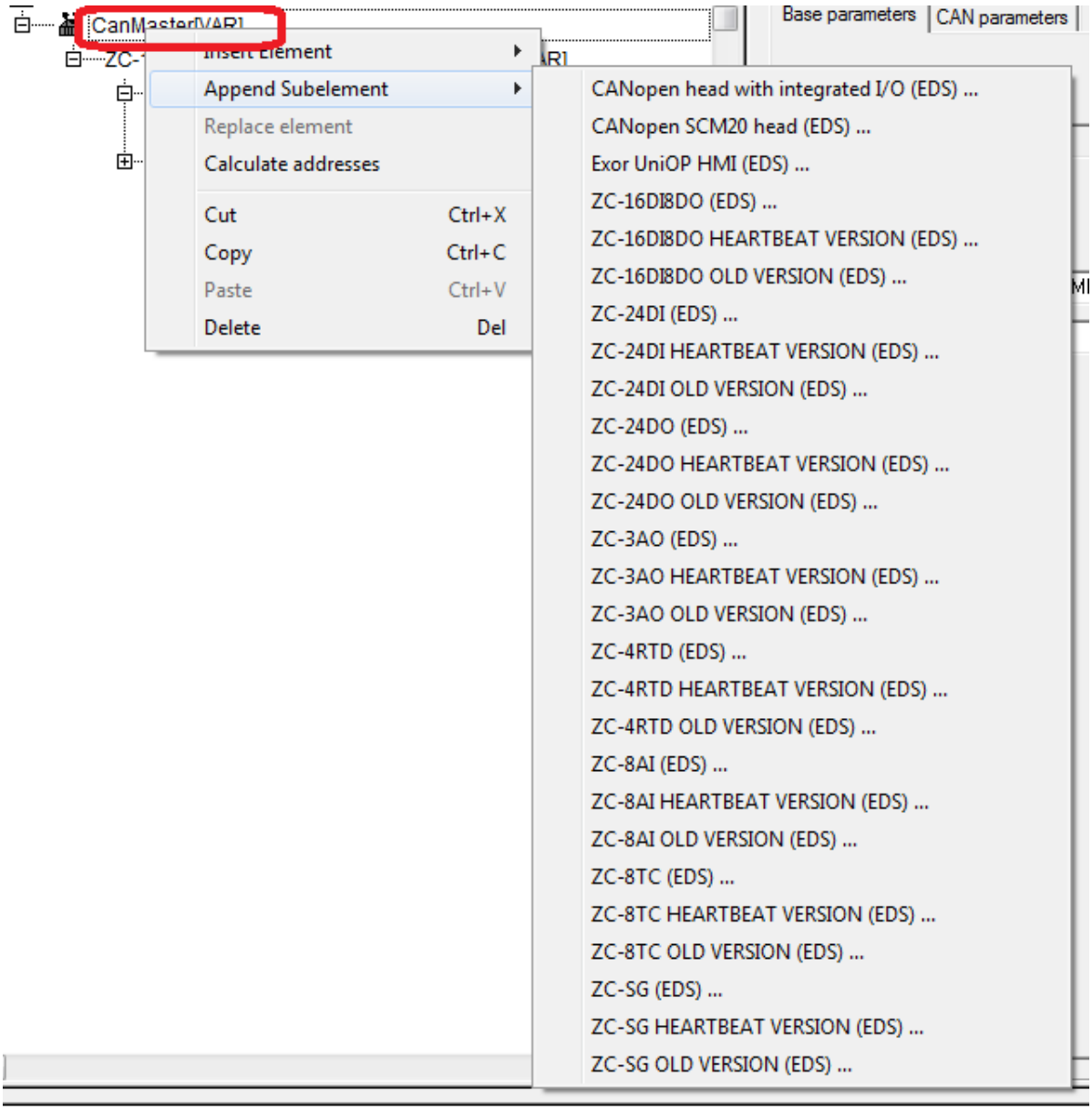


Figure 45 - Configuring the Z-TWS with I/O CANopen – step 11

Configure the corresponding window

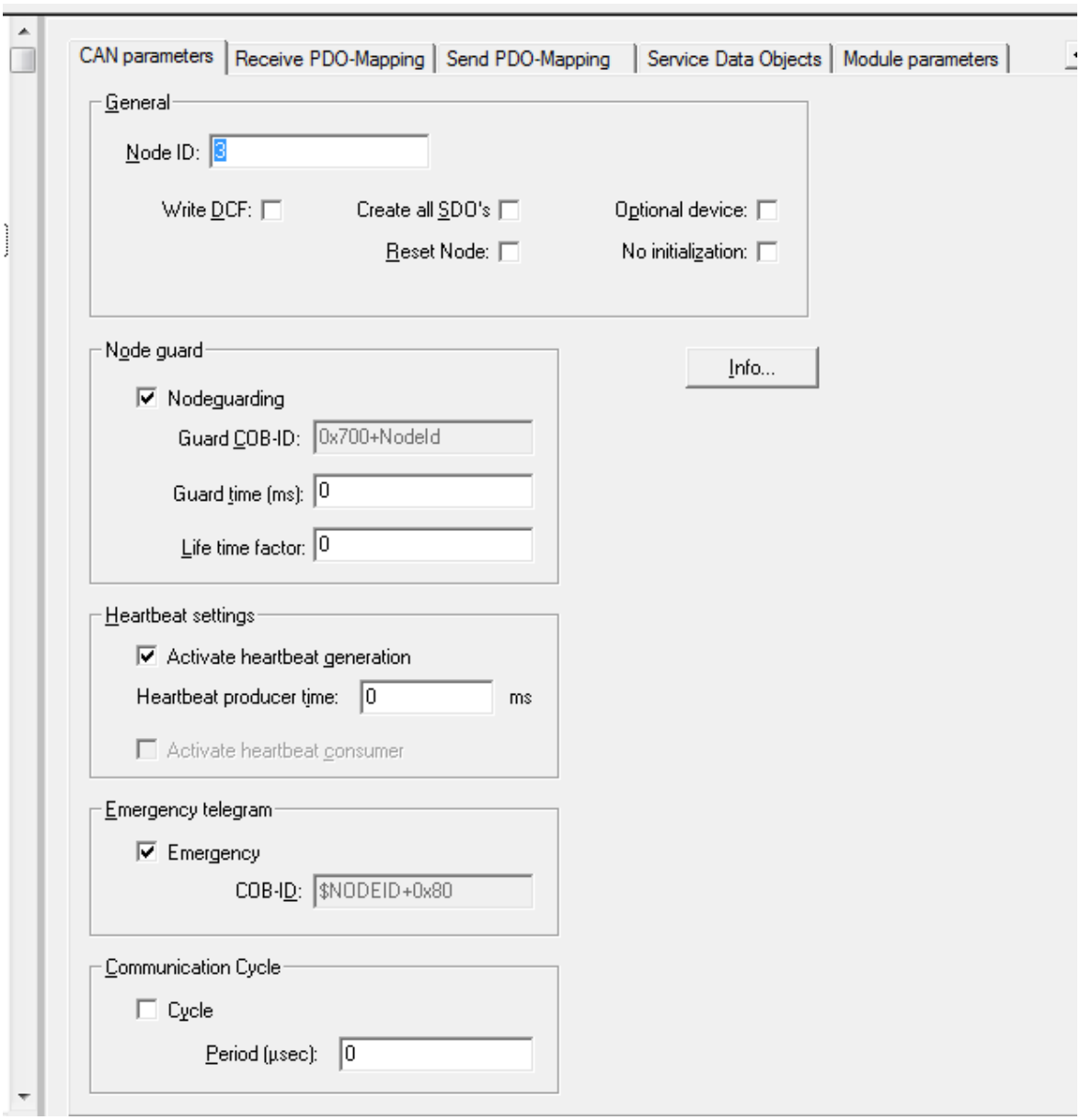


Figure 46 - Configuring the Z-TWS with I/O CANopen – step 12

To download the program, select the Online->Login from menu.

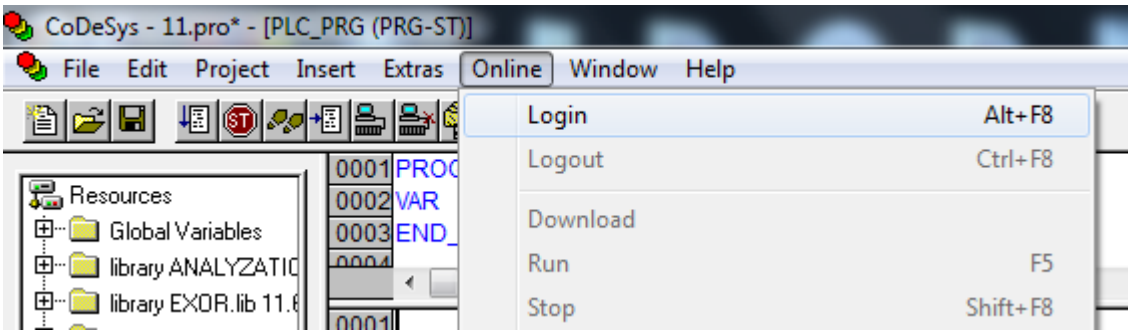


Figure 47 - Configuring the Z-TWS with I/O CANopen – step 13

To run the PLC program, it is necessary to select Online->Run.

How to configure the Web Server: the Web Editor software

The Web editor is a web-server editor for the Z-TWS and allows, mainly, to create easily web pages developed for Z-TWS; moreover, permits to associate directly to the objects the variables defined in the corresponding Z-NET project.

This software can be opened from the software Z-NET, as shown in the following figure.

Figure 48 - Block diagram process to configure the Web Editor

To open the Web Editor software, first it is necessary to open Z-NET and, from Z-NET menu toolbar, select Project->Web Editor

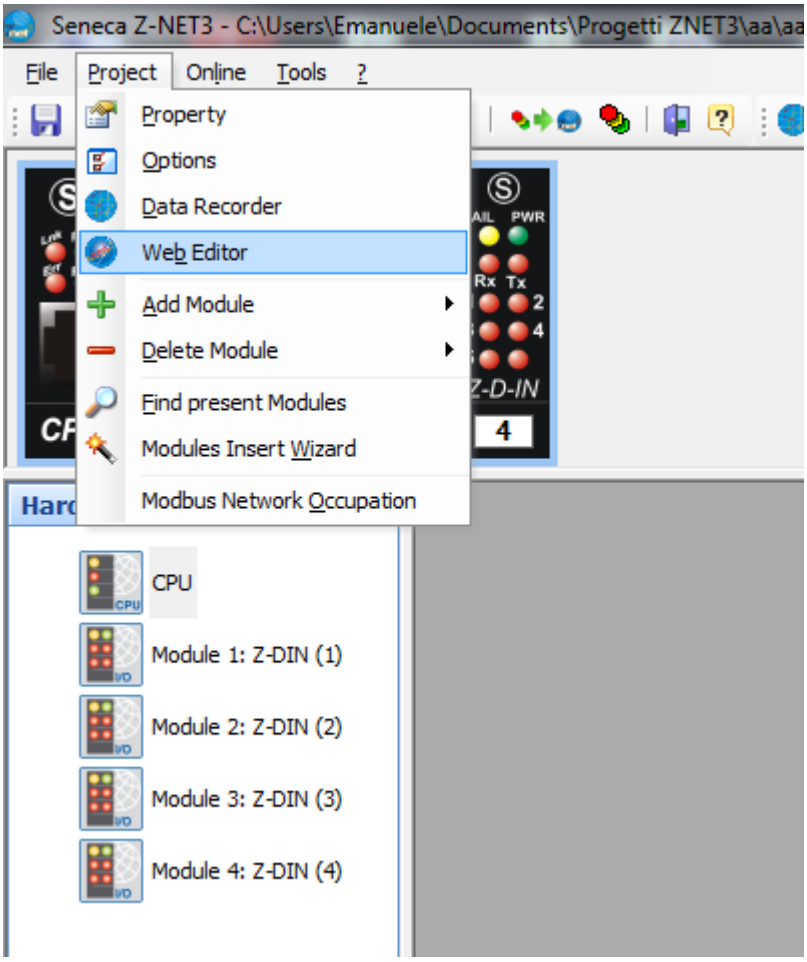


Figure 49 - Starting the Web Editor software from Z-NET

When click on “Web Editor” voice, the program is started and the following window appears: as you can see, more options are available for a Web Editor page.

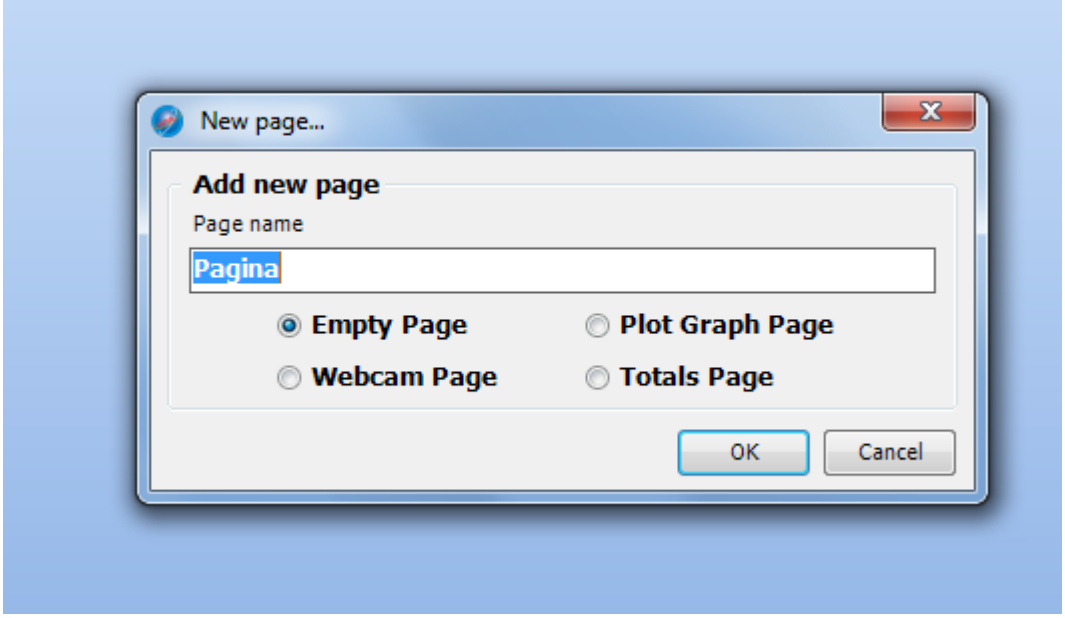


Figure 50 - Window to select the type of the page for Web Editor

For example, select “Empty page”: the Web editor offers at the user some objects that can be used to set a page in a web format, as shown in the following figure.

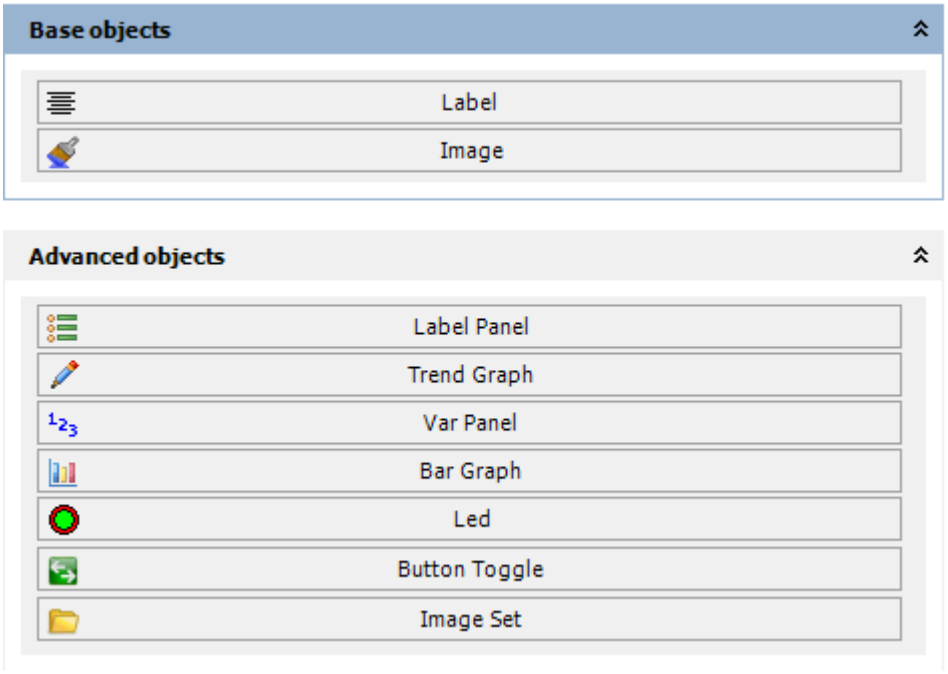


Figure 51 - Objects that can be used to realize a page with Web Editor

The previous objects, described in the following table, can be individually configurated thanks the window “Property” in the right-side of the page, for example: the text, the X-Y position, the colour of the graph, etc…

OBJECT	DESCRIPTION
Label	It allows to insert a simple text label in the page
Image	It allows to insert a simple image in the page
Label panel	Depending on the value of the associated variable, the string configured will be displayed
Trend Graph	It allows to create a graph to trace the desired variables
Var Panel	It allows to load an output or read a variable
Bar graph	It allows to display the value of a variable with a colored bar
Led	It allows to insert a LED in the page, referred to a boolean variable
Button toggle	It allows to insert a button in the page, referred to a boolean variable
Image set	Depending on the value of the associated variable, a different image is displayed

One of more rilevant functionality offered by the previous window consists in the “Var panel” object: it is possible to assign the variables to the Data Acquisition Module automatically, with the following steps:

- open “Web editor” from Z-NET toolbar

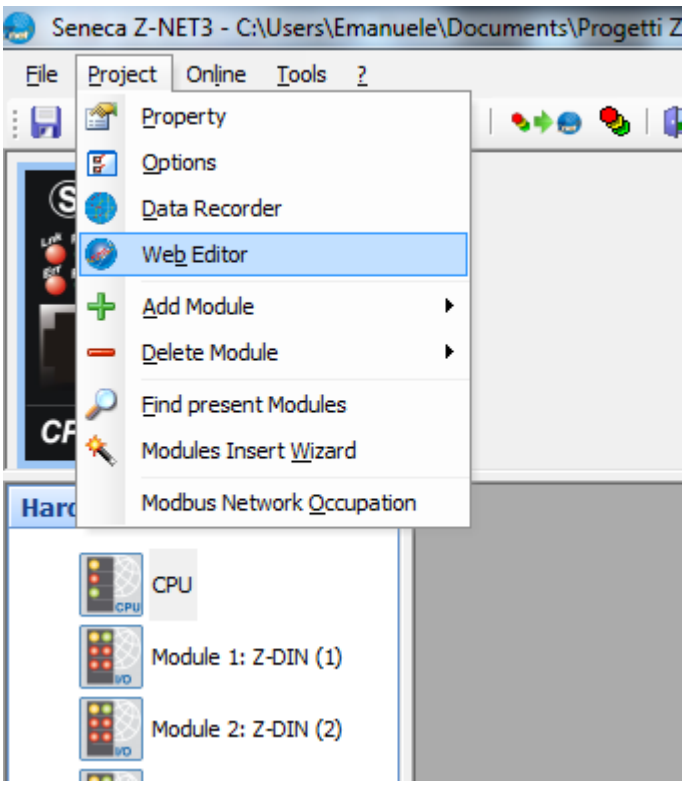


Figure 52 - Starting the Web Editor

- drag the “Var panel” in the left-side window

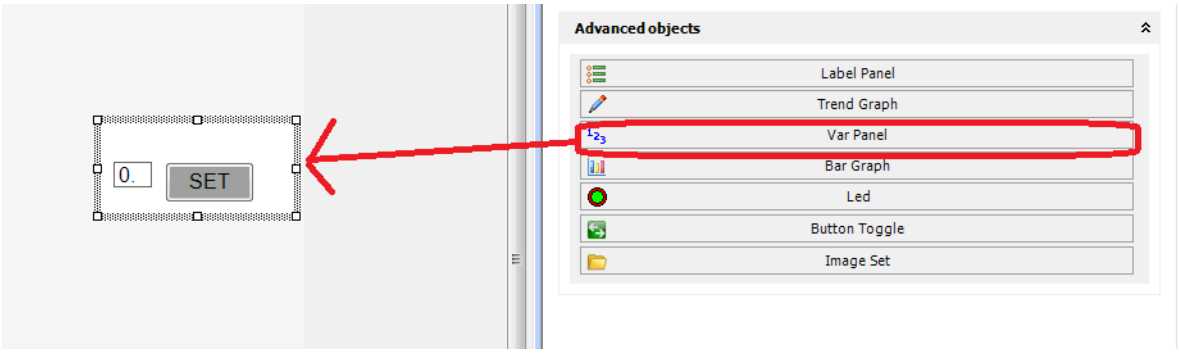


Figure 53 - Drag a “Var panel” object into the page using Web Editor

- double click on the icon of “Var panel”: a “modify control” window appears

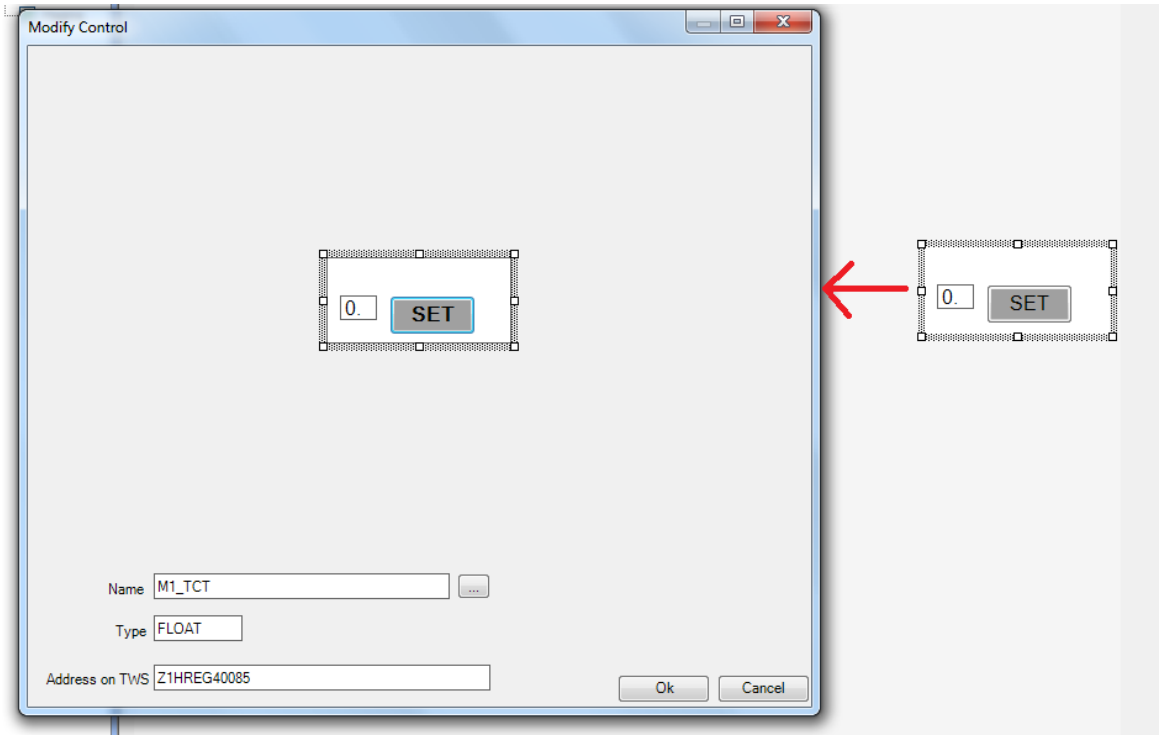


Figure 54 – Modify Control window

- click on the button near “Name” and the “Select variable” window appears

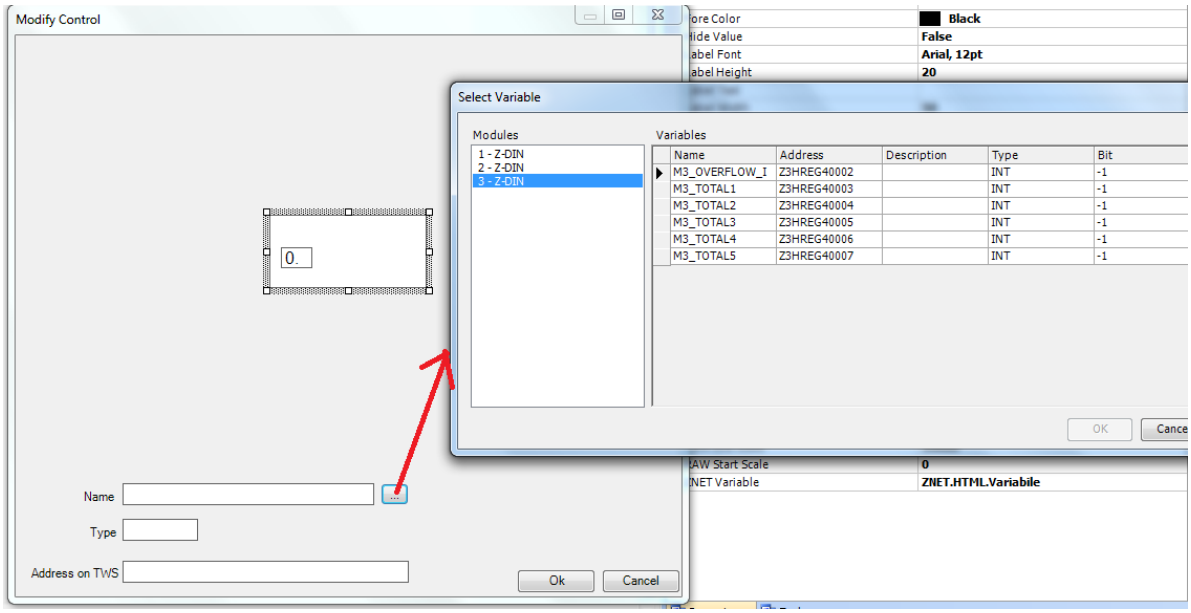


Figure 55 - Assignment of a variable to a “Var panel” object

- for each module selected, the “variables” window shows the name and the corresponding address for each variable that can be assigned to the icon.

Select the desired variable and click on “OK”. To change the default features of this object, select the “Property” window, as for the others objects.

How to configure the Data Recorder

Data recorder is a software that allows to perform the data-logging for each variables; this one is saved into an access data base (mdb). This file can be analyze using the Trend Viewer software, offered in the Seneca Package CD.

Data recorder is a not-free applicative; for more informations, visite the website www.seneca.it or contact Seneca srl.

This software can be opened from the software Z-NET, as shown in the following figure.

Figure 56 - Block diagram process to configure the Data Recorder

When the Data Recorder operates, the PC is configured as Modbus Master: it queries the modules, saves the values and storages them into a Data Base.

NOTE: Seneca provides a variety of RS485-USB optoisolated converters, for example S117P and S117P1; these devices allows to connect safely the Seneca Z-PC Line modules with a PC.

After a configuration of a CPU+I/O project using the Z-NET software, from the Z-NET toolbar, select Project->Data Recorder.

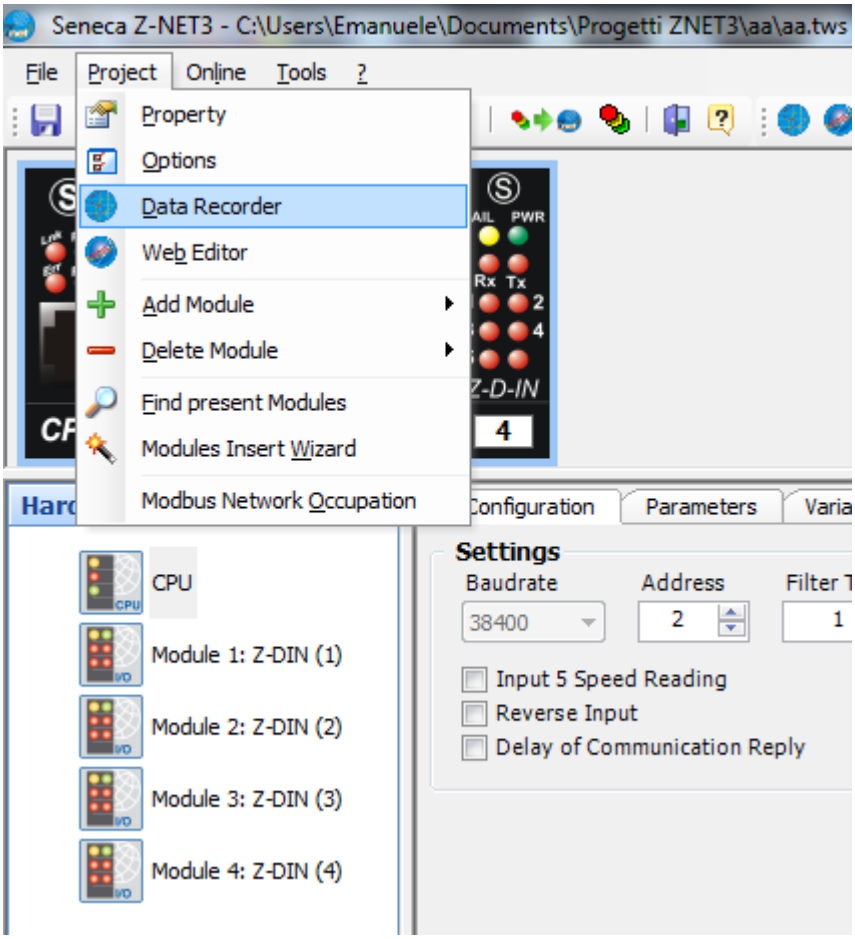


Figure 57 - Configuring the Data Recorder – step 1

At this point, the following window appears.

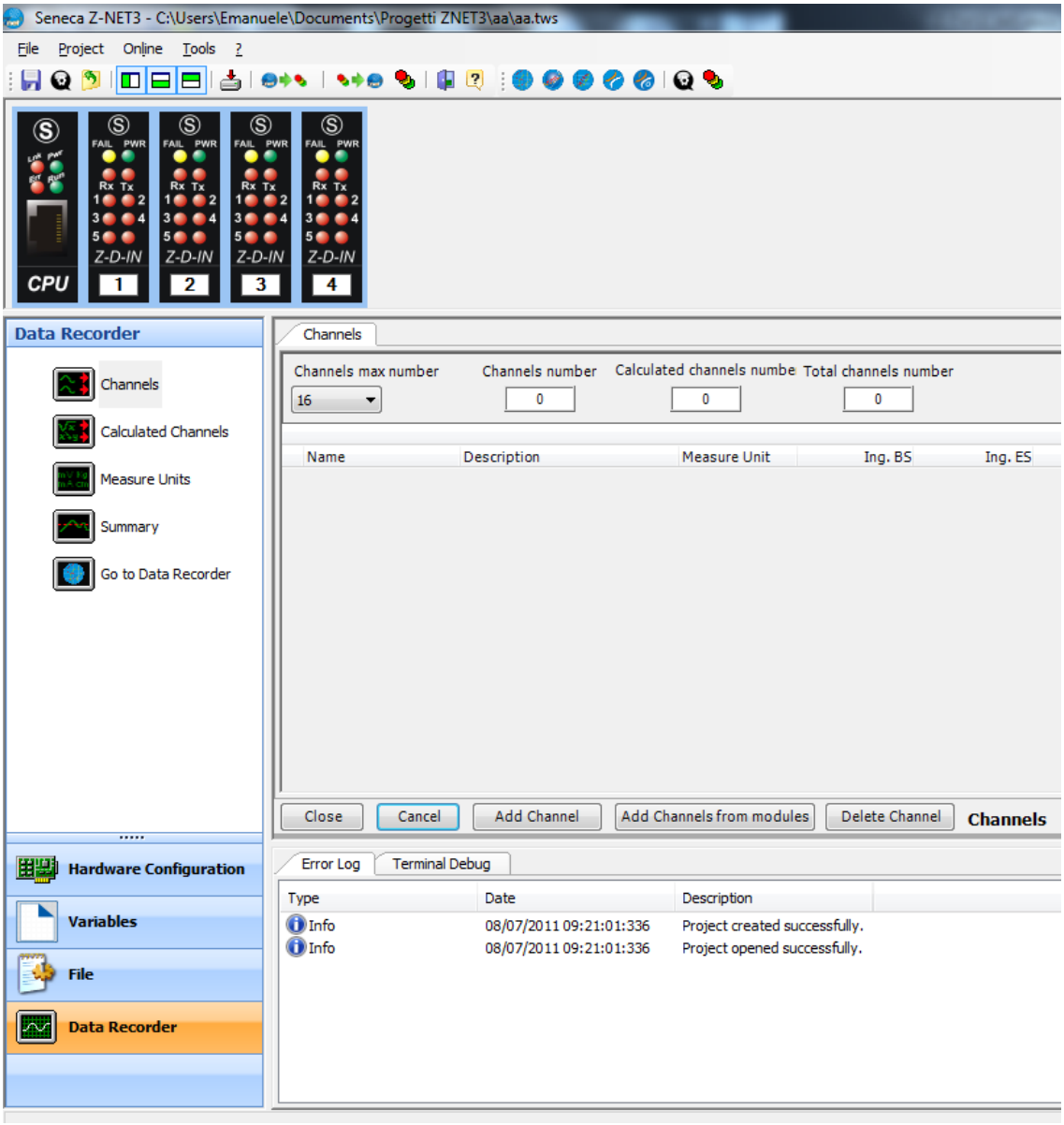


Figure 58 - Configuring the Data Recorder – step 2

Click on “Channels” voice in the left-side window; then click on “Add channel” to add a variable in the Data Recorder project, first choosing the corresponding module and then select one of the variables available for that module. It is possible to analyze more modules and variables at the same time.

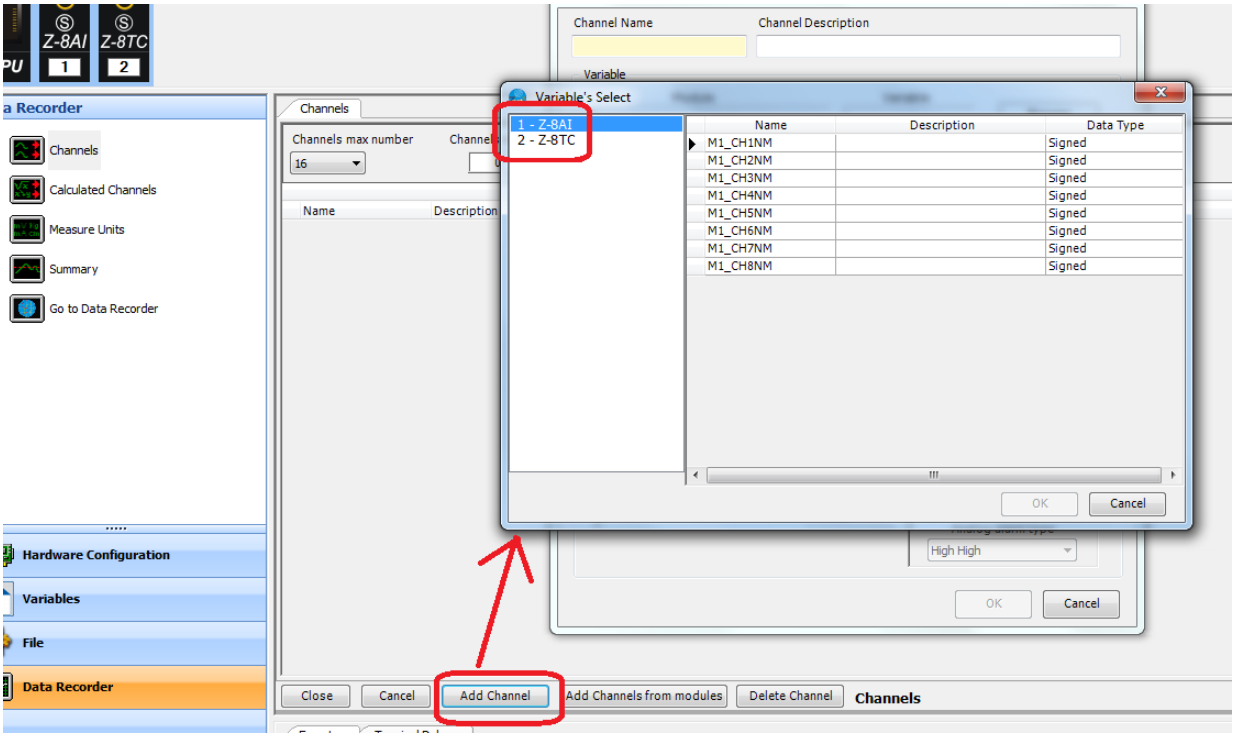


Figure 59 - Configuring the Data Recorder – step 3

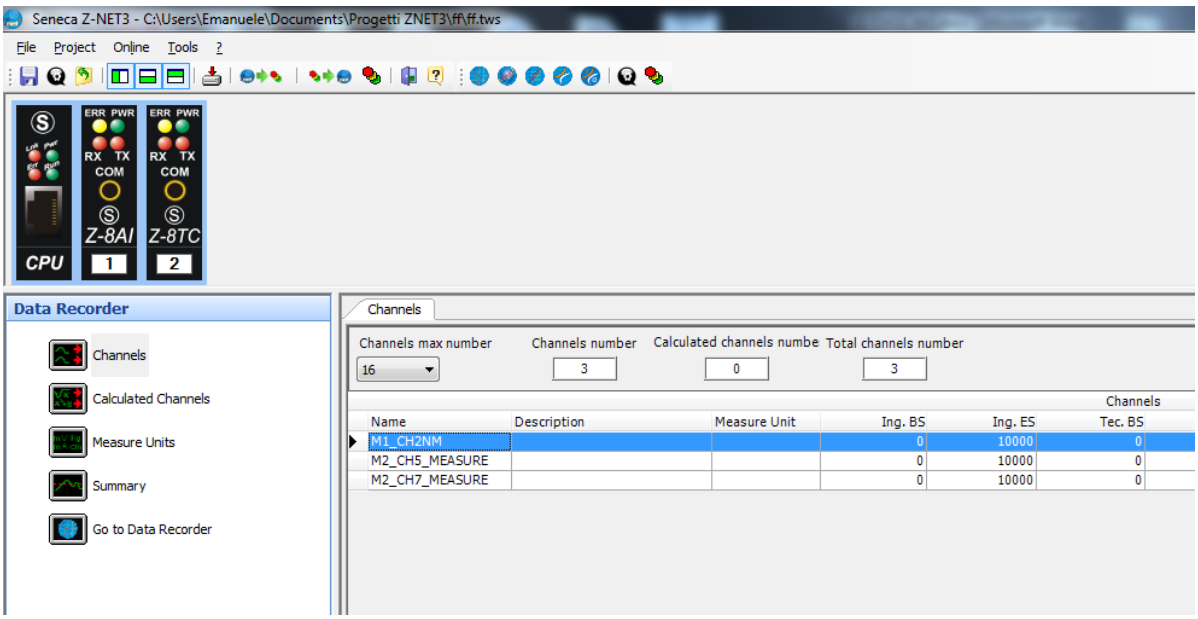


Figure 60 - Configuring the Data Recorder – step 4

Click on the “Calculated Channel” to configure some particular functions that can be associated to each previous variable, for example the alarm or expressions applied to the quantities.

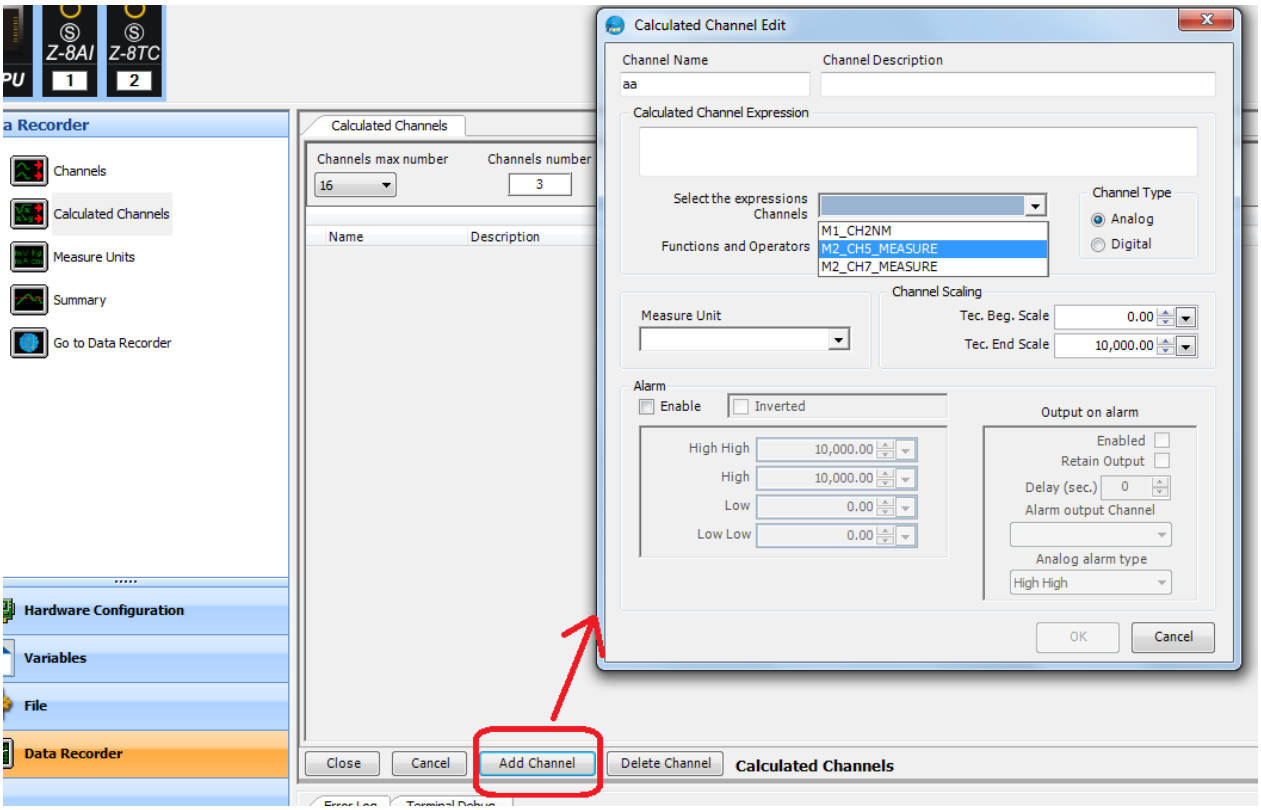


Figure 61 - Configuring the Data Recorder – step 5

Complete this configuration with the “Measure units” and “Summary” voices, and click on “Go to Data Recorder” button: the following window appears.

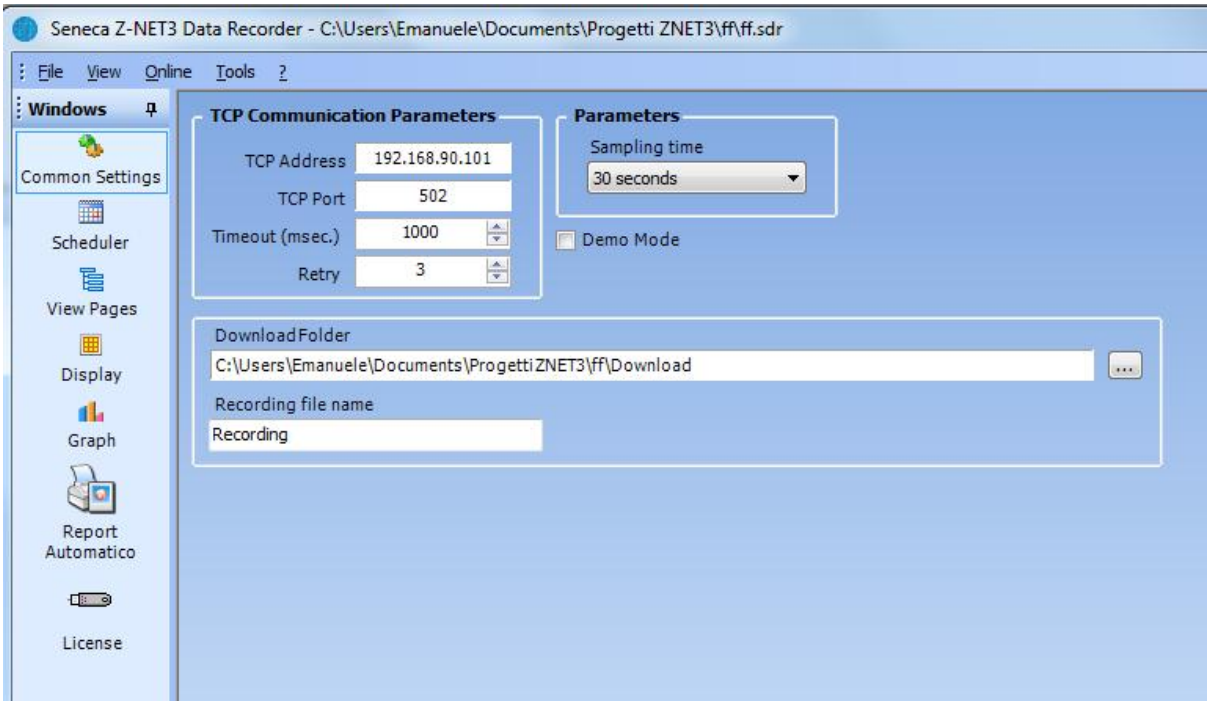


Figure 62 - Configuring the Data Recorder – step 6

Set the TCP communication parameters, sampling time and Download folder voices; after this, click on “Scheduler” to configure the record type (manual or from scheduler) and the other parameters.

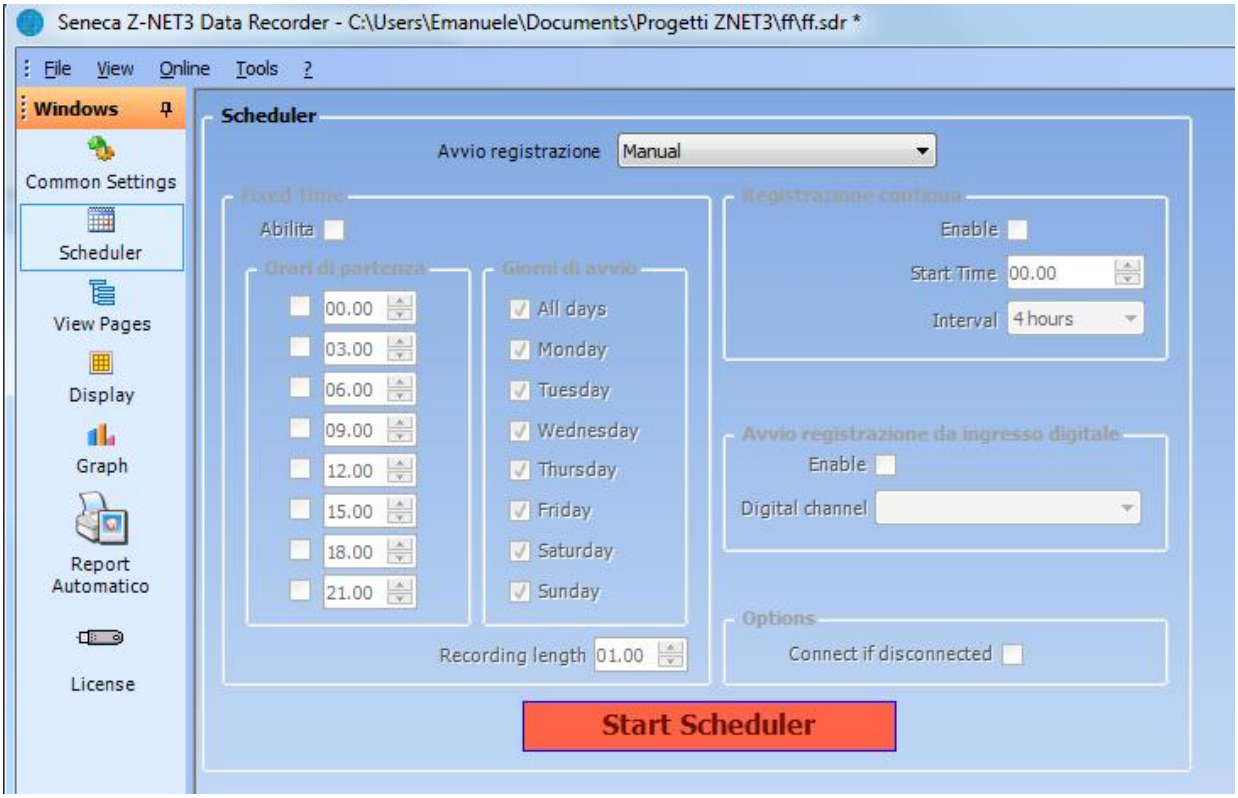


Figure 63 - Configuring the Data Recorder – step 7

Click on the “view pages” to choose the channels that you desire analyze, for each module, and the number of pages. It is possible to add the graph in the report and a table in the report.

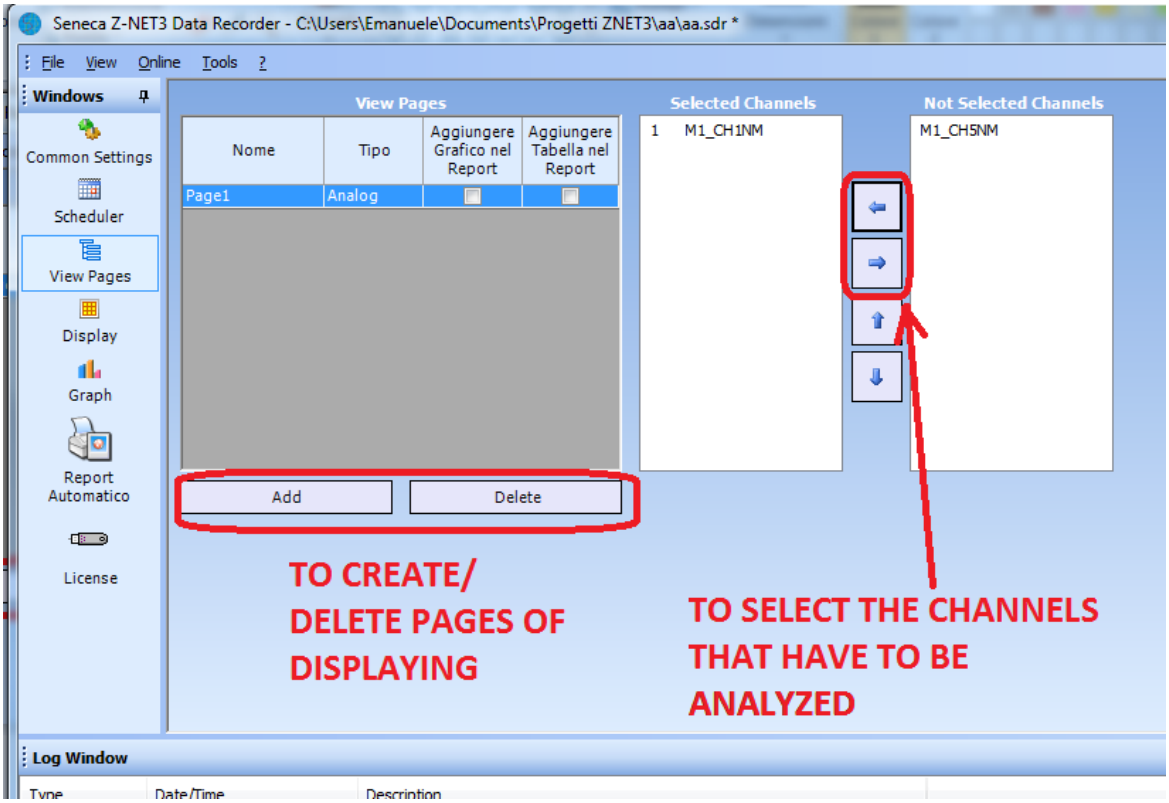


Figure 64 - Configuring the Data Recorder – step 8

After this, click on “Display” and select, from menu, Online->Connect: the instantaneous value/values will be displayed in this window

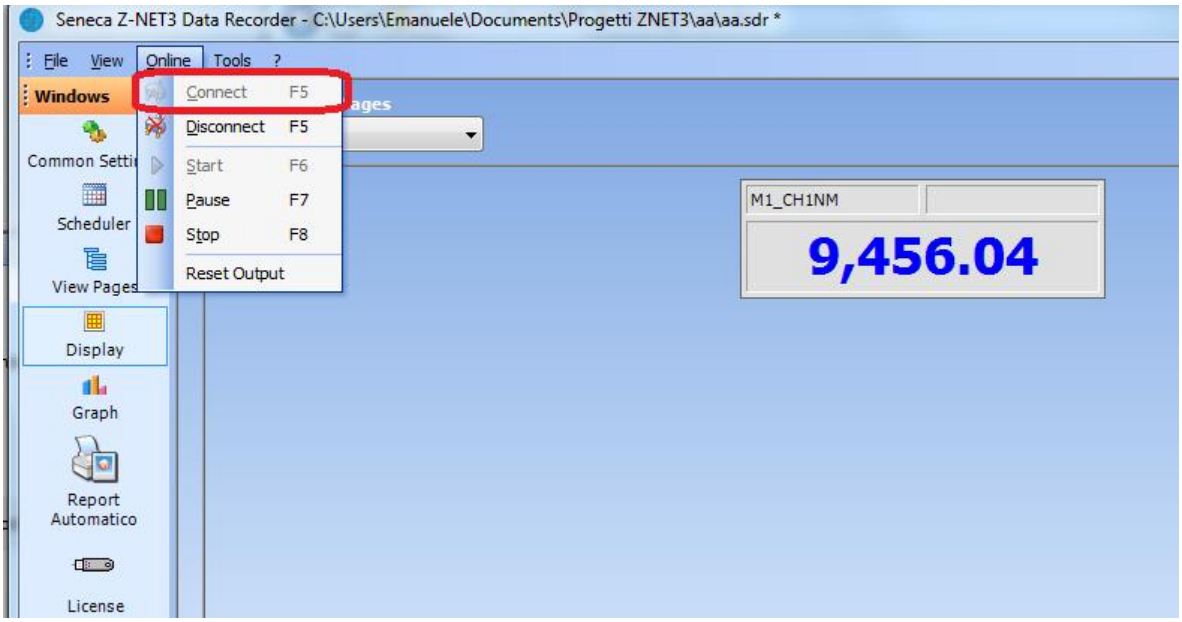


Figure 65 - Configuring the Data Recorder – step 9

while the corresponding time diagram is displayed in the “graph” window

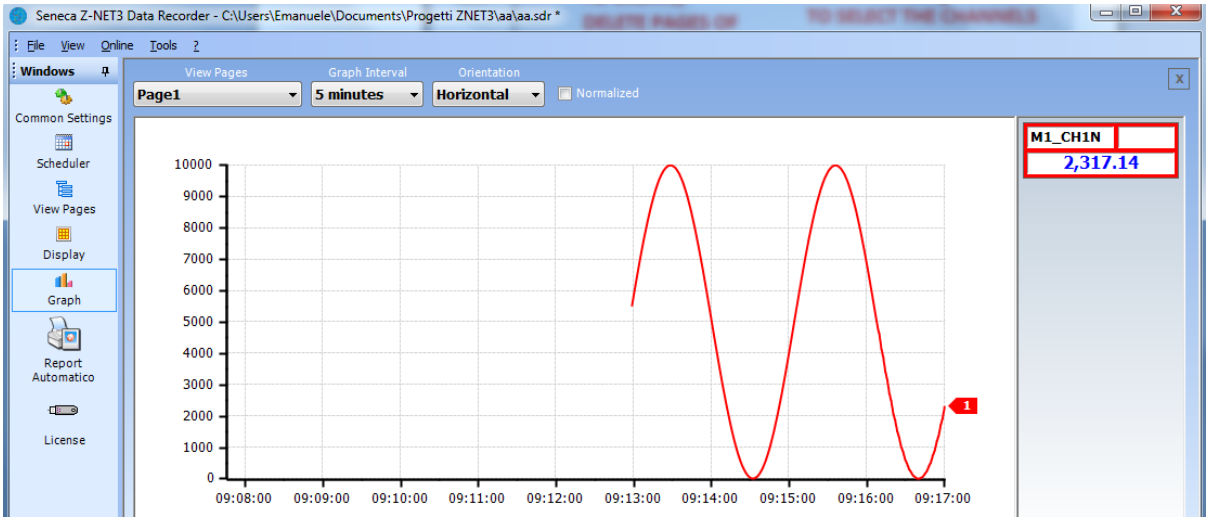


Figure 66 - Configuring the Data Recorder – step 10

The Data Recorder generates an output file (Microsoft® Office® Access data base), in the “download folder”, previous selected. After this, the output file can be analyzed with the Trend Viewer software.

Appendix A: About the PLC.ini file

The PLC.ini file can be easily configured by the Z-NET free software and, in particular, this topic describes the file structure for the Z-TWS.
The first part is about the parameters that are read only at startup:

Syslog=

If you are not using the serial port 0 for debug monitor via the TWS, you can introduce a redirection over ethernet via Syslog, which allows you to send all diagnostic prints to a remote server on the network. In this field is inserted Sylog remote server's IP address.

You must then define the parameters for setting the serial ports, those that are not indicated are free:

#Uart parameter section (starting from 0)

For port 0, the parameters are as follows

Mode0=0

The modes are allowed 0 = RS232, 1 = 485 3 = 485 master and slave (external)

Channel0=-1

DO NOT CHANGE: indicates a serial line

Baud_Rate0=38400

Indicates the communication speed of the port for which the values are: 4800, 9600, 19200, 38400, 57600, 115200 baud

Parity0=N

Is possible to add, sending each character, a bit that indicates if the number of bits equal to 1 character is odd or even. This is useful for detecting transmission errors due to noise on the line, as the receiver replays the count and compares the result to the parity bit received. Menu options are none, even, odd.

Length0=8

Is possible to choose if you send or receive all 8 bits of character, or 7.

Stop0=1

The stop bits are equal to 1 bits that are appended to each character sent. These are used to mark the end of the bits of the character and the values are 1 and 2.

Half_Duplex0=N

NO CHANGE

The parameters are configured according to the requirements for all other ports that are not Free:

Mode1=1

Channel1=-1

Baud_Rate1=9600

Parity1=N

Length1=8

Stop1=1

Half_Duplex1=N

Mode2=1

Channel2=-1

Baud_Rate2=9600

Parity2=N

Length2=8

Stop2=1

Half_Duplex2=N

Mode3=3

Channel3=-1

Baud_Rate3=9600

Parity3=N

Length3=8

Stop3=1

Half_Duplex3=N

Similarly for all ports scheduled are not free; it is essential to distribute the addresses of the slaves among the communication ports and overlaps are not allowed.

Serial2=1

FirstSlave2=41

LastSlave2=150

Delay2=0

RequestTimeout2=20

RequestNtry2=0

Serial3=2

FirstSlave3=151

LastSlave3=200

Delay3=0

RequestTimeout3=20

RequestNtry3=0

Serial4=3

FirstSlave4=201

LastSlave4=254

Delay4=0

RequestTimeout4=20

RequestNtry4=0

Finally, therefore, are defined blocks optimized communication (up to 1024). These blocks indicate to the MODBUS protocol such as, how many and how often the records should be asked of a particular I / O. The protocol is optimized to handle the reading of a group of user-defined variables. After defining the groups are updated periodically with the interval set (MMS). Any requests for variables belonging to the groups by the three main sources (PLCs, TWS, DAM) does not give rise to new communication sessions, but are met instantaneously by taking the data from memory.
If writing a variable belonging to a group, it involves a writing session and if it succeeds, the new value is forced in the memory block belonging.

The first parameter contains the address of the register to be read, this addressing is done through a string like this:

Zxx<TYPE>yyyyy

where:

xx = number of slave MODBUS(1 - 255)

yyyyy = number of selected object

TYPE = type of required data, can be assume the following:

OUTP = digital output ex. OUTP0001

INP1 = digital input ex. INP10002

IREG3 = input registers ex. IREG30004

HREG4 = input registers ex. HREG40022

In normal operation with modules of the I / O SENECA is used the request HREG4.
The format of communication blocks is as follows:

BlockBase0=Z2HREG40002	start
BlockLength0=2	lenght
BlockSamplingRate0=1000	refresh in ms

BlockBase1=Z3HREG40020

BlockLength1=4

BlockSamplingRate1=500

BlockBase2=Z4HREG40002

BlockLength2=2

BlockSamplingRate2=50

NOTE: The Modbus registers are 16 bits so if, for example, you want to read 3 registers, BlockLength parameter must contain the value 6, if you want to read the 4 parameter must contain the value 8, and so on.
In the example above, the first block requires reading (every 1000mSec) of register 40002 of the expansion module with address 2; the second block requires reading (every 500msec) registers 40020 and 40021 of the expansion module with address 3.

Appendix B: Seneca libraries for CoDeSys

IMPORTANT!

The file system are managed by the OS File System, the number of files that can be opened at the same time is limited, as well the number of files that can be saved in a same directory and in the root.

It is important to consider that on a PLC the flash has a limited cycles of writing.

To avoid decrease in performances (like a PLC watchdog reset), it is not recommended to perform multiple operations in the same file.

In particular it's recommended to avoid:

- ***to send a file through FTP while this file is open***
- ***to create a high number of files in a folder or a high number of folders***
- ***using different threads that perform multiple accesses to a same file***

The performances of the system depend on the I/O modules that have to be managed.

To improve the system performances, it is important to consider that the access to files affects heavily in the cycle time of PLC. So, it is necessary to optimize the accesses to the file system in the PLC program.

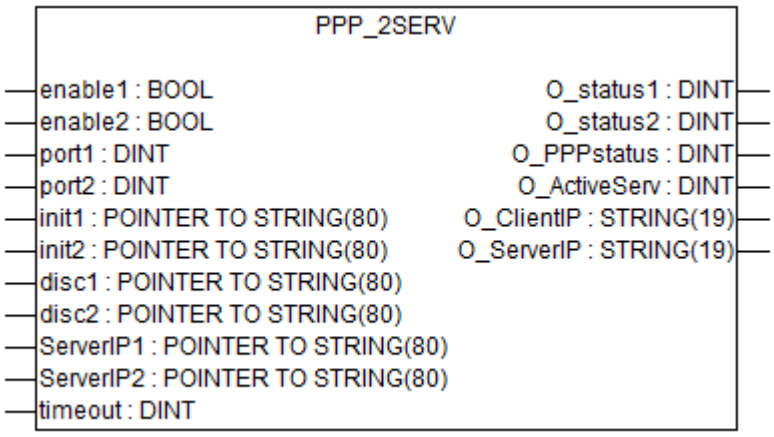
Library: TWS_MISC.lib

Name: **PPP_2Serv**

Description:

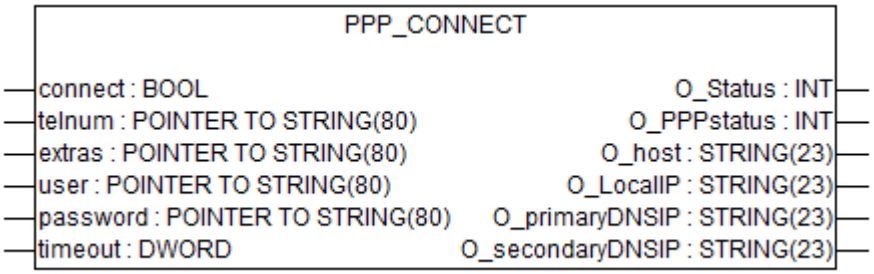
Enable PPP server on 2 modems.

The PPP server is automatically started on the MODEM which receives a RING.



Name: **PPP_connect**

Description: PPP connection control.



Name: **PPP_GETSMS**

Description:

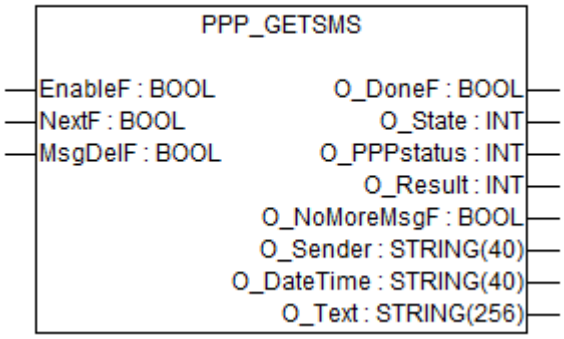
Get SMS from GSM modem SIM card.

Rising edge of "EnableF Input" starts operations.

As soon the first SMS has been fetched, the "DoneF Ouput" becomes true and message text is available at "Text Output".

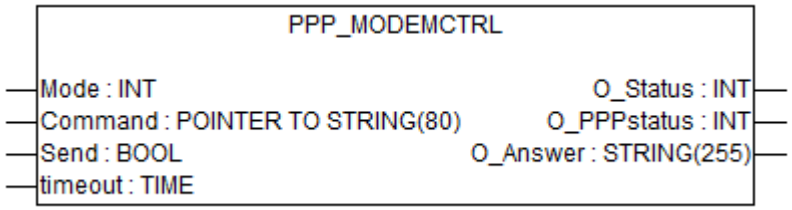
At this point, a rising edge on "NextF Input" causes another SMS to be fetched.

"DoneF Output" becomes false and is rose again as soon the second SMS has been fetched.



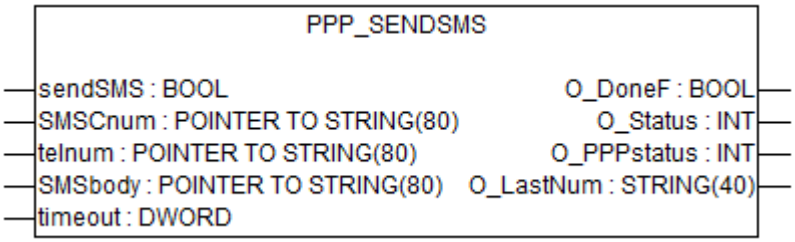
Name: **PPP_ModemCtrl**

Description: allow various functions of control on Modem



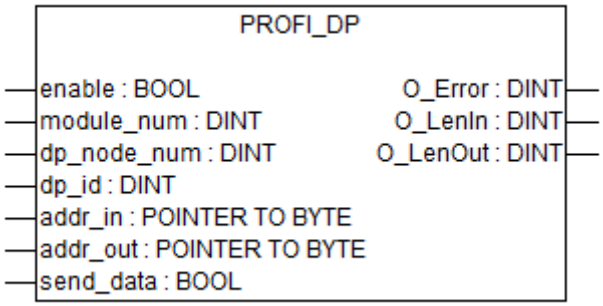
Name: **PPP_SendSMS**

Description: PPP control for SMS send on GSM modem



Name: **Profi_DP**

Description: Profibus DP light driver

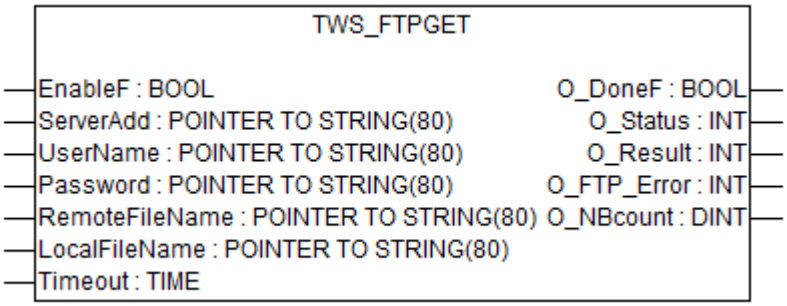


Name: **TWS_FTPget**

Description:

Connects to an FTP server and retrieves a file.

This FB posts a "get request" to the thread which performs the actual transfer.

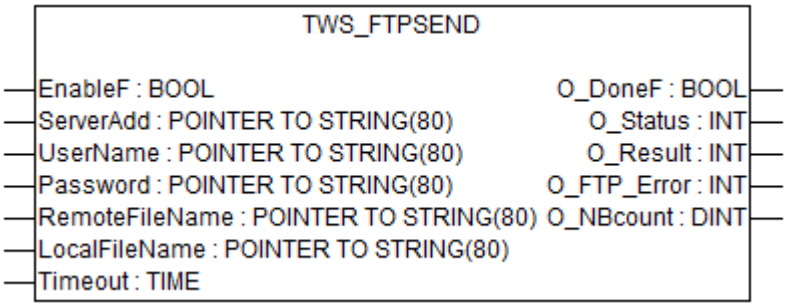


Name: **TWS_FTPsend**

Description:

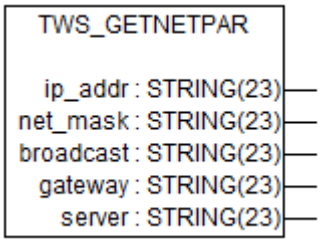
Connects to an FTP server and sends a file.

This FB posts a "send request" to the thread which performs the actual transfer.



Name: **TWS_GetNetPar**

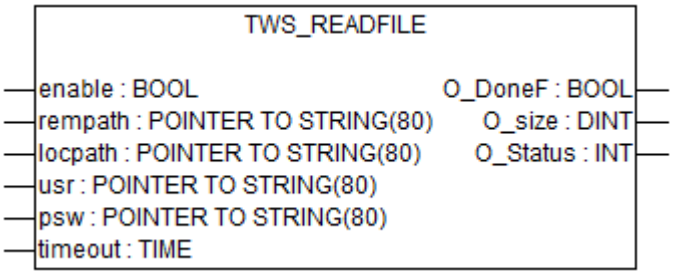
Description: Get Network Parameters



Name: **TWS_IsGtwRegistered**

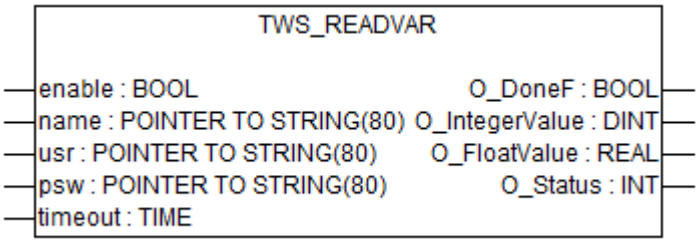
Name: **TWS_readFile**

Description: Read one file from remote TWS to local TWS



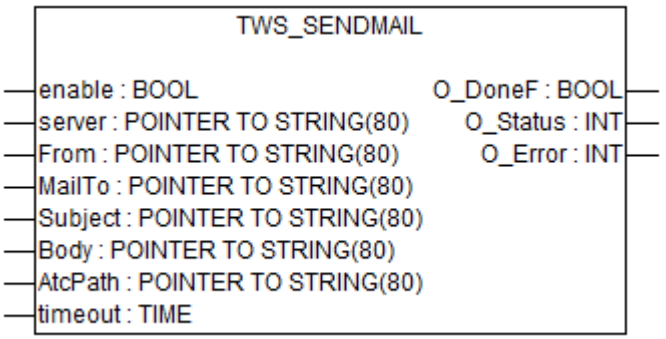
Name: **TWS_readVar**

Description: Read one variable from local or remote TWS



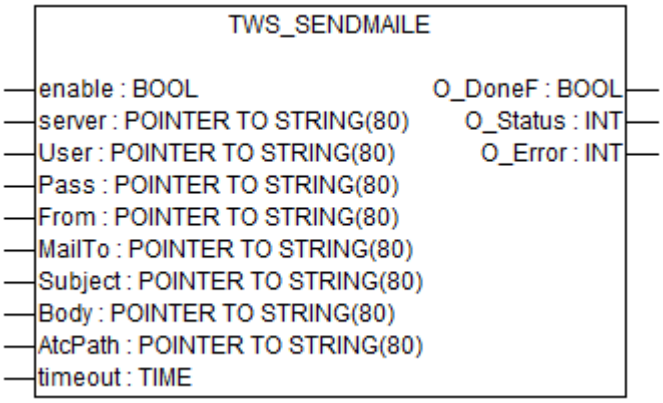
Name: **TWS_sendMail**

Description: Send a E-Mail message



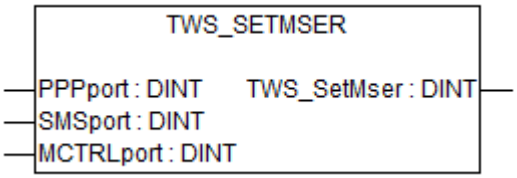
Name: **TWS_sendMailE**

Description: Send a E-Mail message


TWS_SENDMAILE
enable : BOOL
server : POINTER TO STRING(80)
User : POINTER TO STRING(80)
Pass : POINTER TO STRING(80)
From : POINTER TO STRING(80)
MailTo : POINTER TO STRING(80)
Subject : POINTER TO STRING(80)
Body : POINTER TO STRING(80)
AtcPath : POINTER TO STRING(80)
timeout : TIME
O_DoneF : BOOL
O_Status : INT
O_Error : INT

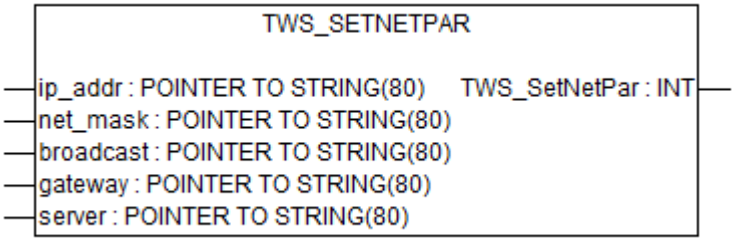
Name: **TWS_SetMser**

Description: Sets serial ports used by PPP SMS and MCTRL.



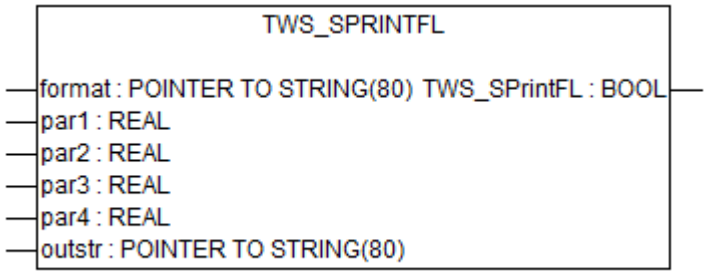
Name: **TWS_SetNetPar**

Description: Set Network Parameters


TWS_SETNETPAR
ip_addr : POINTER TO STRING(80)
net_mask : POINTER TO STRING(80)
broadcast : POINTER TO STRING(80)
gateway : POINTER TO STRING(80)
server : POINTER TO STRING(80)
TWS_SetNetPar : INT

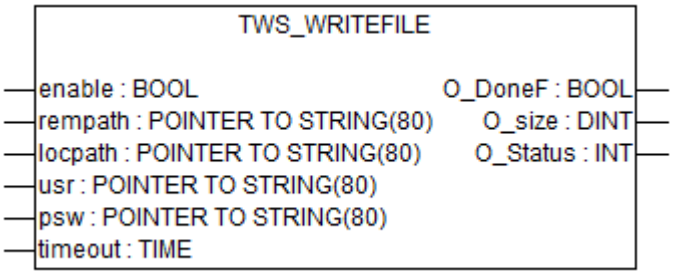
Name: **TWS_SPrintFL**

Description: create a message variable with C like formatting capabilities



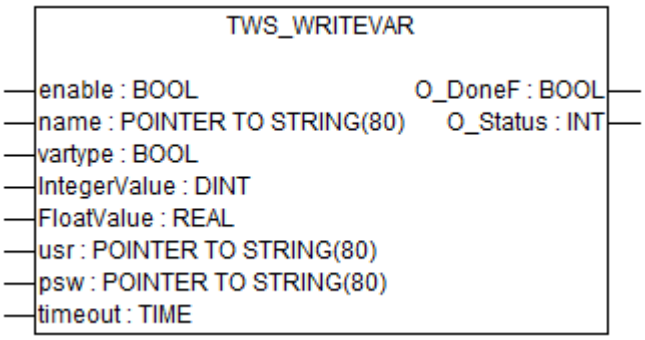
Name: **TWS_writeFile**

Description: write one file form local to remote TWS



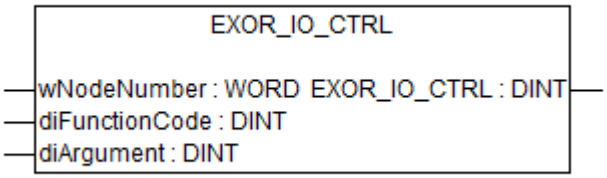
Name: **TWS_writeVar**

Description: write one variable to local or remote TWS



Library: EXOR.lib

Name: **EXOR_IO_CTRL**



Library: EXOR_CAN.lib

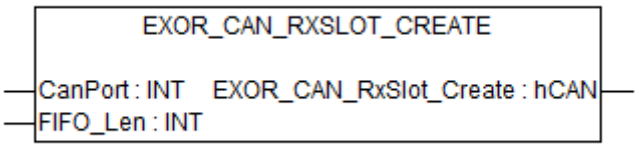
Name: **EXOR_CAN_RxSlot_Create**

Description:

This function creates an Rx Slot. An Rx Slot is a place (a buffer) where received CAN messages are deposited. The parameter "FIFO_Len" specify how many CAN messages the Rx Slot can contain. Rx Slot is a buffer FIFO style (First In First Out). Application program calls "EXOR_CAN_RxSlot_GetPacket()" function to unload one message at a time from this FIFO.

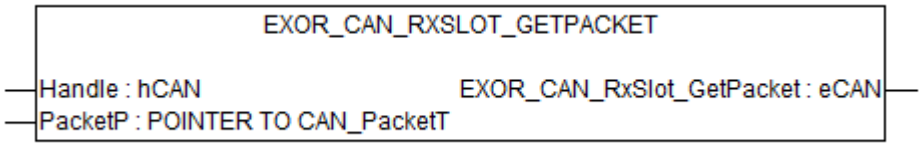
How "FIFO_Len" parameter should be calculated ? First the execution time of the full application program should be measured / estimated, then the quantity of CAN messages which can be received in the Rx Slot in this time should be computed.

Parameter "CanPort" selects the CAN Port, if your hardware has only one CAN Port, then specify zero for this parameter. After the Rx Slot is created, it is configured for receiving no message at all. To select which CAN messages have to be received by an Rx Slot, the function "EXOR_CAN_RxSlot_SetIdAndMask()" must be used.



Name: **EXOR_CAN_RxSlot_GetPacket**

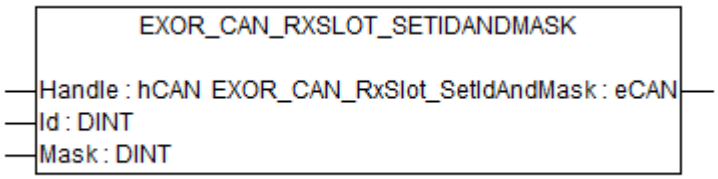
Description: This function unloads one message from the specified RxSlot.



Name: **EXOR_CAN_RxSlot_SetIdAndMask**

Description:

This function is used after creating an RxSlot with "EXOR_CAN_RxSlot_Create()" function. Application program can call "EXOR_CAN_RxSlot_SetIdAndMask()" function each time it wants to change which CAN messages has to be received on an RxSlot. If same Identifier/Mask is used in more than one RxSlot then the last created RxSlot receives it. Mask parameter gives you the possibility to receive multiple Identifiers in the same RxSlot. (Bits: A '0' value means 'don't care' or accept a '0' or '1' for that bit position. A '1' value means that the incoming bit value 'must-match' identically to the corresponding bit of the specified "Id") To completely disable the reception of CAN messages on an RxSlot, application program should call this function with Id = -1 and Mask = -1.



Name: **EXOR_CAN_SendPacket**

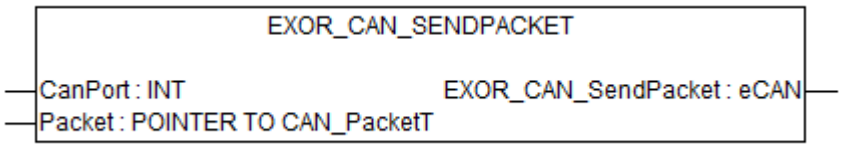
Description:

WARNING: There is a 64 messages buffer FIFO, messages are deposited in this FIFO and are transmitted by interrupt service procedures as soon there are no higher precedence messages

in the CAN bus. If FIFO gets full because application program calls this function at a rate higher than CAN bus throughput, then error _CAN_FIFO_FULL is returned.

Return:_Return Value is _CAN_OK if message has been queued in the FIFO buffer (or directly transmitted if FIFO is empty).

*R*eturn Value is _CAN_FIFO_FULL if FIFO is full.



Library: EXOR_CAN2.lib

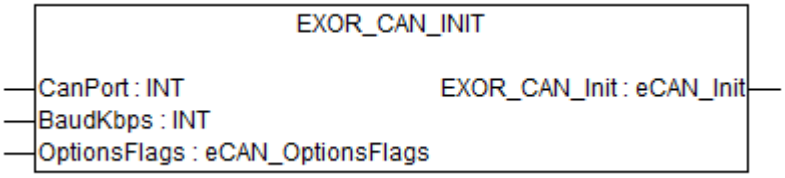
Name: **EXOR_CAN_Init**

Description: Initialize specified CanPort.

WARNING:If specified CanPort is already used in a CanMaster board in PLC Configuration, then the CanPort is already initialized and EXOR_CAN_Init MUST NOT be called.

Return: WARNING: For Error Codes of this function, please see eCAN_Init DataType.

_CANDRV_OK if successful, otherwise



Name: **EXOR_CAN_rxSlot_Create**

Description:

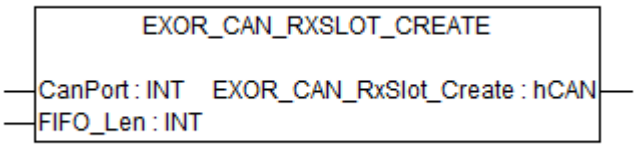
WARNING: Each time this function is called, an RxSlot is created and some memory depending on FIFO_Len parameter is allocated for it. For this reason, application program must not continuosly create Rx Slots, instead application program should create the needed quantity of Rx Slots at the startup and then use these Rx Slots.

Purpose: This function creates an Rx Slot. An Rx Slot is a place (a buffer) where received CAN messages are deposited. The parameter "FIFO_Len" specify how many CAN messages the Rx Slot can contain. Rx Slot is a buffer FIFO style (First In First Out). Application program calls "EXOR_CAN_RxSlot_GetPacket()" function to unload one message at a time from this FIFO.

How "FIFO_Len" parameter should be calculated ? First the execution time of the full application program should be measured / estimated, then the quantity of CAN messages which can be received in the Rx Slot in this time should be computed.

Parameter "CanPort" selects the CAN Port, if your hardware has only one CAN Port, then specify zero for this parameter. After the Rx Slot is created, it is configured for receiving no message at all. To select which CAN messages have to be received by an Rx Slot, the function "EXOR_CAN_RxSlot_SetIdAndMask()" must be used.

Return: An handle to the created RxSlot is returned. The returned handle has to be used to reference the RxSlot when calling "EXOR_CAN_RxSlot_SetIdAndMask()" and "EXOR_CAN_RxSlot_GetPacket()" functions. If no memory is available to create the RxSlot, then NULL (zero) is returned.



Name: **EXOR_CAN_rxSlot_GetPacket**

Description: This function unloads one message from the specified RxSlot.

Entry:

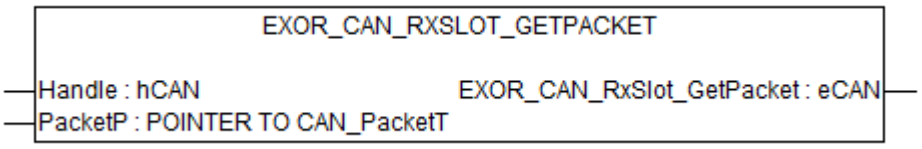
Handle: Handle of a previously created RxSlot

PacketP: Pointer to the buffer where unloaded packet will be written.

Return:

Return value is _CAN_NO_DATA if no message exists in the RxSlot.

Return value is _CAN_OK if one message has been unloaded from RxSlot.



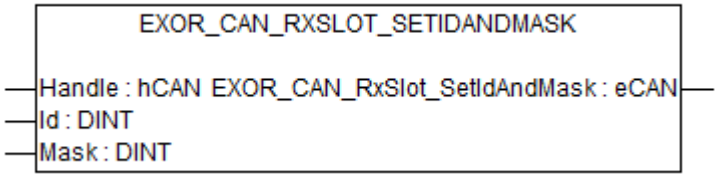
Name: **EXOR_CAN_RxSlot_SetIdAndMask**

Description:

This function is used after creating an RxSlot with "EXOR_CAN_RxSlot_Create()" function.

Application program can call "EXOR_CAN_RxSlot_SetIdAndMask()" function each time it

Identifier/Mask is used in more than one RxSlot then the last created RxSlot receives it. Mask parameter gives you the possibility to receive multiple Identifiers in the same RxSlot. (Bits: A '0' value means 'don't care' or accept a '0' or '1' for that bit position. A '1' value means that the incoming bit value 'must-match' identically to the corresponding bit of the specified "Id") To completely disable the reception of CAN messages on an RxSlot, application program should call this function with Id = -1 and Mask = -1.



Name: **EXOR_CAN_RxSlot_SendPacket**

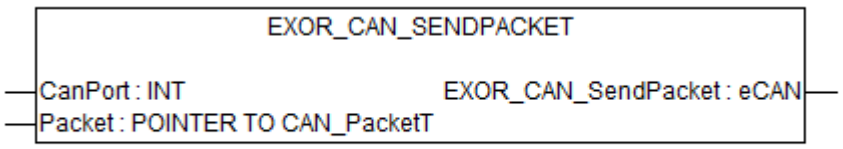
Description:

WARNING: There is a 64 messages buffer FIFO, messages are deposited in this FIFO and are transmitted by interrupt service procedures as soon there are no higher precedence messages in the CAN bus. If FIFO gets full because application program calls this function at a rate higher than CAN bus throughput, then error _CAN_FIFO_FULL is returned.

Return:

Return Value is _CAN_OK if message has been queued in the FIFO buffer (or directly transmitted if FIFO is empty).

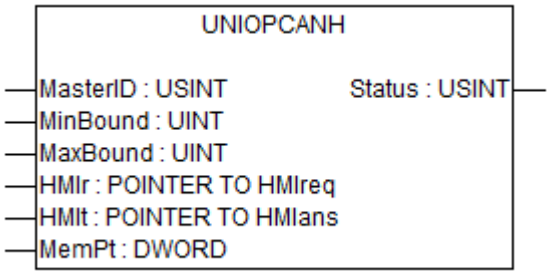
Return Value is _CAN_FIFO_FULL if FIFO is full.



Library: EXOR_Canh.lib

Name: **UniopCANH**

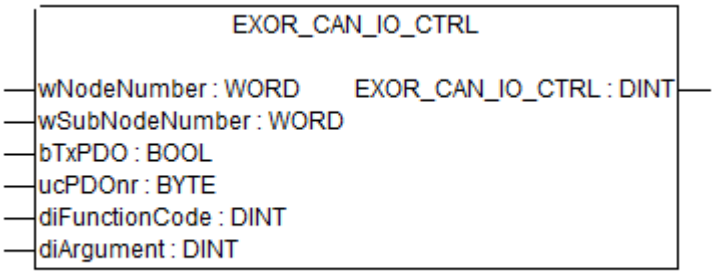
Description:



Library: EXOR_CANopen.lib

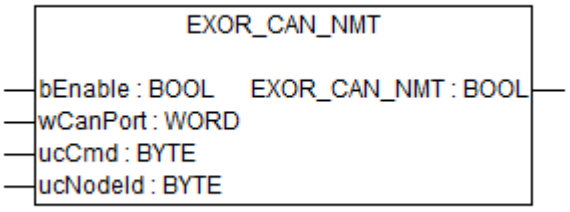
Name: **EXOR_CAN_IO_CTRL**

Description:



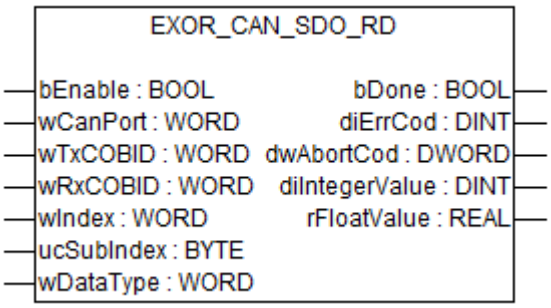
Name: **EXOR_CAN_NMT**

Description:



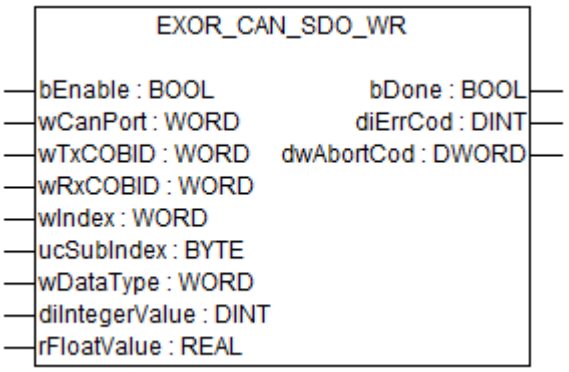
Name: **EXOR_CAN_SDO_RD**

Description:



Name: **EXOR_CAN_SDO_WR**

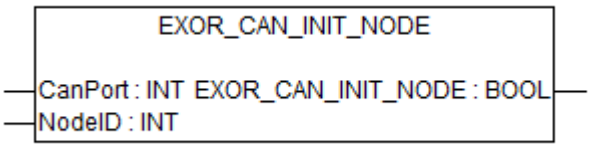
Description:



Library: EXOR_CANopen2.lib

Name: **EXOR_CAN_INIT_NODE**

Description:



Name: **EXOR_CAN_SET_RXONLY_F**

Description:

IF TRUE, then CAN Master will only act for receiving from configured Slaves Nodes,

CAN Master will not do NMT and will not send PDO to configured Slaves Nodes.

IF FALSE, then CAN Master will act normally.

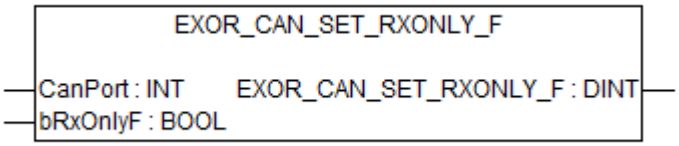
When switching from TRUE to FALSE, the normal Start Up procedure will be executed,

i.e. the Slaves Nodes will be initialized and then they will be started (NMT Start) and Node Guarded.

Return:

Return Value is _CAN_OK if CanPort is valid, otherwise _CAN_INVALID_HANDLE is returned

(see EXOR_CAN.LIB for return codes).



Library: EXOR_CANopenBlockTransfer.lib

Name: **EXOR_CAN_BT_CLIENT_CHECK**

Description:

IEC program after calling one time EXOR_CAN_BT_CLIENT_DL or EXOR_CAN_BT_CLIENT_UL, then it will repeatedly call this function to check if download operation has been completed.

Entry: ResPtr: this must be the address (ADR() operator) of a EXOR_CAN_BT_CLIENT_CHECK_T structure, where result data will be written. Please see the comments embedded in the EXOR_CAN_BT_CLIENT_CHECK_T Data Type definition.

Exit: Result data will be written at the address specified by ResPtr parameter.

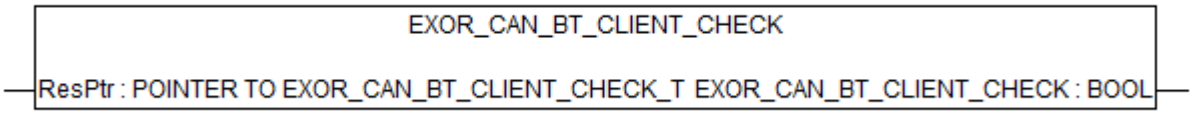
WARNING: Please note that result data is written to ResPtr ONLY when function returns TRUE.

Return:

TRUE when operation has been completed (both in case of success and in case of failure)

FALSE if operation hasn't yet completed.

WARNING: Please note that TRUE will be returned only ONE time after the operation has been completed.



Name: **EXOR_CAN_BT_CLIENT_DL**

Description:

WARNING: Both client and server can do only one operation at a time, i.e. they can do a download or an upload.

They CAN'T do both a download and an upload at the sametime.

They CAN'T more then one upload or download at the sametime.

Purpose: IEC program calls this function to request a download.

IEC program calls this function one time and then repeatedly calls EXOR_CAN_BT_CLIENT_CHECK() to check if download operation has been completed.

Entry:

Note: CANopen standard defines dwServerRxId as 16#600+NodeId

Note: CANopen standard defines dwServerTxId as 16#580+NodeId

Return:

Please see comments embedded in the EXOR_CAN_BT_ResultCodesT defintion.

NOTE:

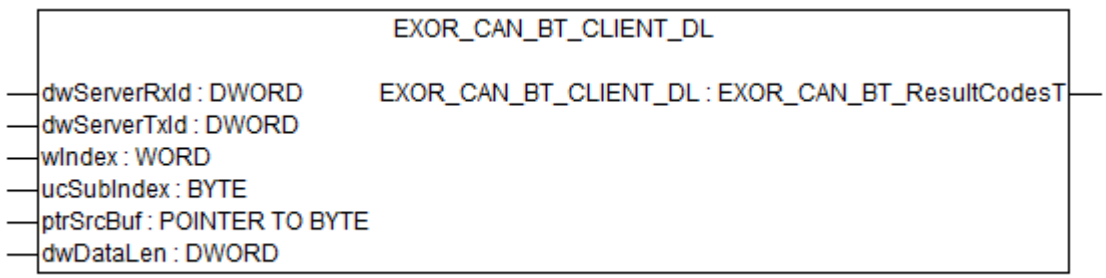
Data is transferred while IEC program runs (by Interrupt Service Procedures).

Data is read directly from the specified address ptrSrcBuf while the transfer proceeds.

This means that IEC program must ensure that Source Data does NOT change until the transfer has been completed.

This can be easily accomplished by IEC program by passing to this function a buffer which is a copy of the source data.

This way the original source data, but the copy buffer will not.



Name: **EXOR_CAN_BT_CLIENT_UL**

Description:

WARNING: Both client and server can do only one operation at a time, i.e. they can do a download or an upload.

They CAN'T do both a download and an upload at the sametime.

They CAN'T more than one upload or download at the sametime.

Purpose: IEC program calls this function to request an upload.

IEC program calls this function one time and then repeatedly calls EXOR_CAN_BT_CLIENT_CHECK() to check if upload operation has been completed.

Entry:

Note: CANopen standard defines dwServerRxId as 16#600+NodeId

Note: CANopen standard defines dwServerTxId as 16#580+NodeId

bAcceptSmallerSizeF: if TRUE then the upload operation will perform also if Server returns less data than the one specified with dwDataLen parameter.

If FALSE then the upload operation will perform only if Server returns exactly the specified datalen.

Return:

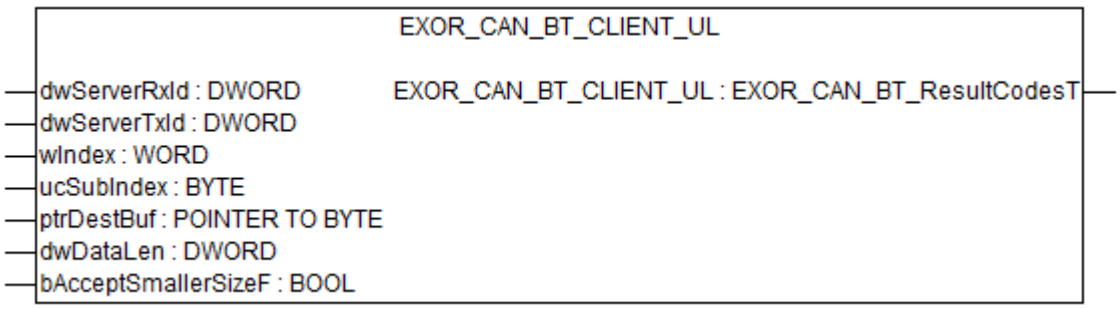
Please see comments embedded in the EXOR_CAN_BT_ResultCodesT defintion.

NOTE:

Data is transferred while IEC program runs (by Interrupt Service Procedures).

Data is written directly from the specified address ptrSrcBuf while the transfer proceeds.

This means that IEC program must not use the Data until the transfer has been completed.



Name: **EXOR_CAN_BT_SERVER_DL**

Description:

WARNING: Both client and server can do only one operation at a time, i.e. they can do a download or an upload.

They CAN'T do both a download and an upload at the sametime.

They CAN'T more then one upload or download at the sametime.

Purpose: IEC program must repeatedly call this function for each download it wants to accept.

If a download at the specified wIndex / ucSubIndex has been successfully received, then data will be copied to the specified address (ptrDestBuf) and for the specified dwDataLen.

If data length specified by the Client differs from the one specified here (dwDataLen parameter), then Server refuses the download and returns an Abort Message to the client.

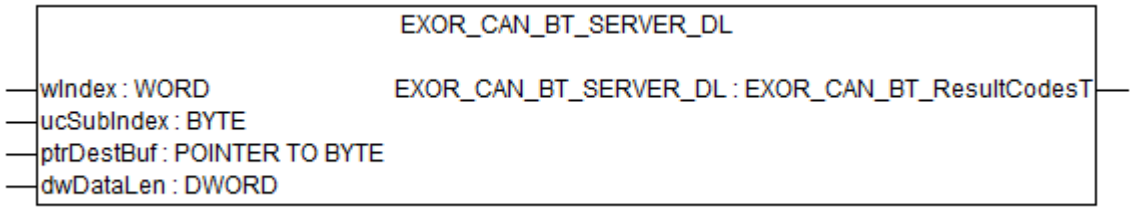
NOTE:

When this function is called for the first time for a given wIndex / ucSubIndex, then a buffer of dwDataLen size is allocated and associated to the specified wIndex / ucSubIndex.

Data is received while IEC program runs (by Interrupt Service Procedures) and is stored in this buffer.

When IEC program calls this function then data is copied from this buffer to ptrDestBuf address.

This ensures coherence of data, because data seen by IEC only changes when calling this function.



Name: **EXOR_CAN_BT_SERVER_SET_CFG**

Description:

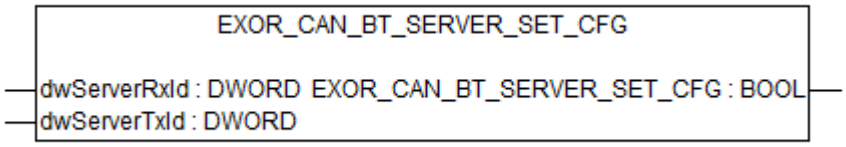
IEC program must call this function only one time at the startup and before any calls to EXOR_CAN_BT_SERVER_DL or EXOR_CAN_BT_SERVER_UL functions.

The Server will work at the specified RX / TX Identifiers.

IEC program can call this function again to change the RX / TX Identifiers, but this is not a common case.

Note: CANopen standard defines dwServerRxId as 16#600+NodeId

Note: CANopen standard defines dwServerTxId as 16#580+NodeId



Name: **EXOR_CAN_BT_SERVER_UL**

Description:

WARNING: Both client and server can do only one operation at a time, i.e. they can do a download or an upload.

They CAN'T do both a download and an upload at the sametime.

They CAN'T more then one upload or download at the sametime.

Purpose: IEC program must call this function for each upload it wants to prepare.

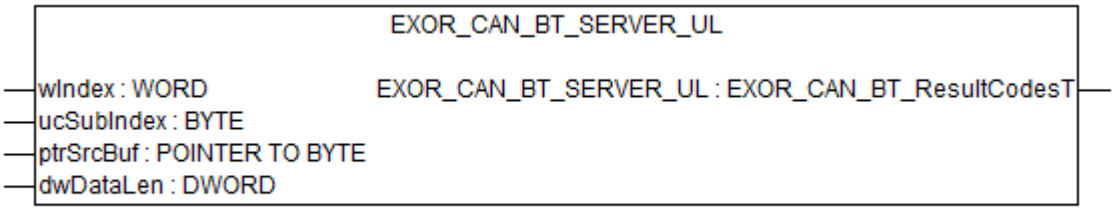
IEC program must call this function to update upload data.

IEC program can call this function continously or only when data changes.

Entry:

ptrSrcBuf: Pointer to source data.

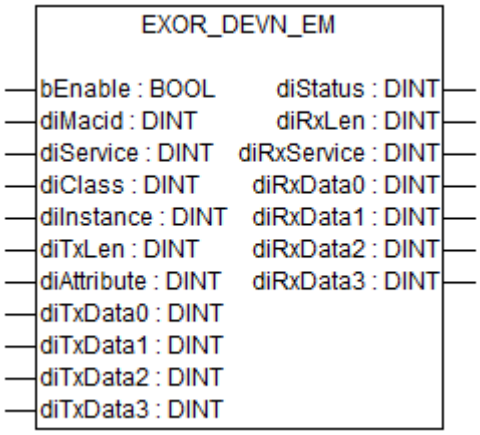
dwDataLen: Length of source data.



Library: EXOR_DeviceNET.lib

Name: **EXOR_DEVN_EM**

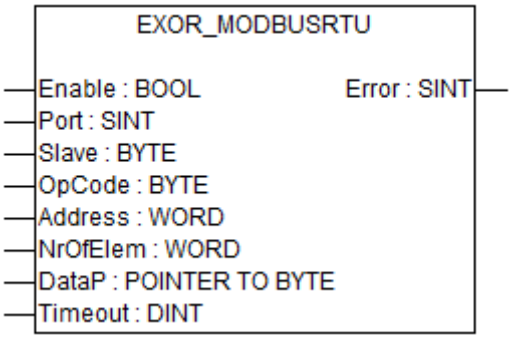
Description:



Library: EXOR_Modbus_RTU.lib

Name: **EXOR_ModbusRTU**

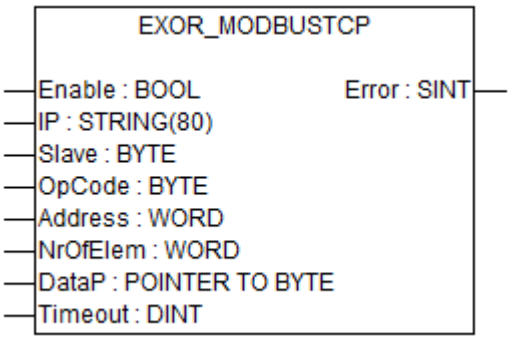
Description: Modbus master protocol engine



Library: EXOR_Modbus_TCP.lib

Name: **EXOR_ModbusTCP**

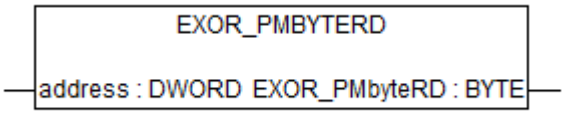
Description: Modbus master protocol engine



Library: EXOR_PMbus.lib

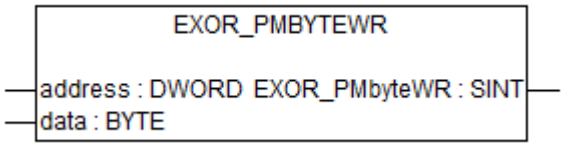
Name: **EXOR_PMbyteRD**

Description: read one byte from selected slave at specified address



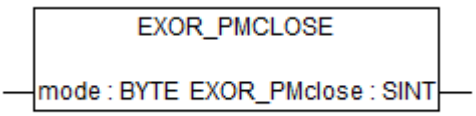
Name: **EXOR_PMbyteWR**

Description: write one byte to selected slave at specified address

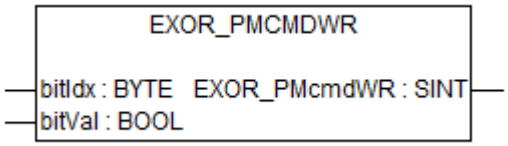


Name: **EXOR_PMclose**

Description: stop the I/O bus

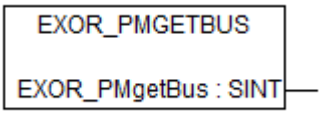


Description: write one byte to command register of selected slave



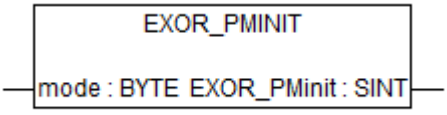
Name: **EXOR_PMgetBus**

Description: wait until Bus becomes free (in case of other tasks were accessing the bus



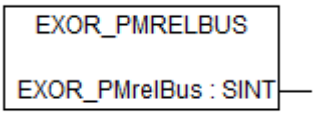
Name: **EXOR_PMinit**

Description: initialize the I/O bus



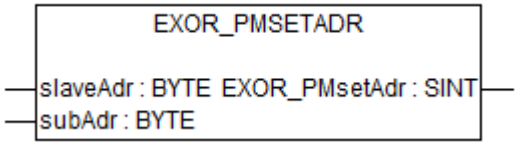
Name: **EXOR_PMrelBus**

Description: release the use of the Bus



Name: **EXOR_PMsetAdr**

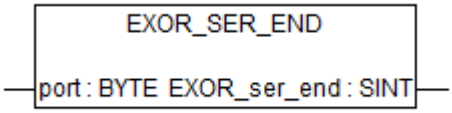
Description: select one slave on the bus for future use



Library: EXOR_Serial.lib

Name: **EXOR_ser_end**

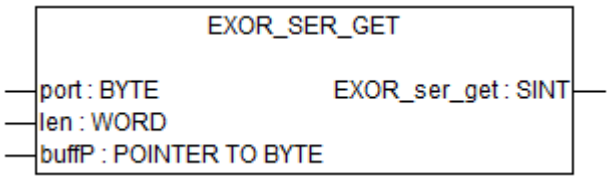
Description: make one serial port free for other uses; result of the operation 0=no errors -1=port was not in use



Name: **EXOR_ser_get**

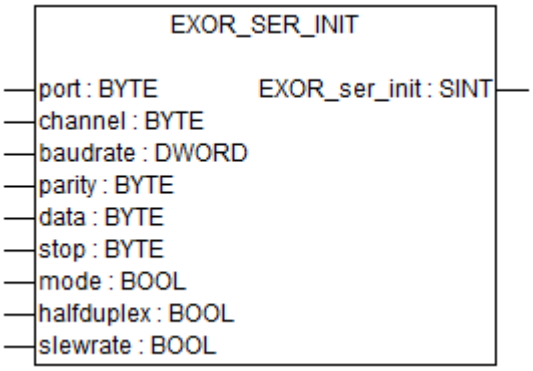
Description: read some characters from one serial port; result of the operation 0=no errors -1=port not in use -2=not enough chars

ATTENTION: dimension of array must be large enough to contain number of characters specified in LEN parameter



Name: **EXOR_ser_init**

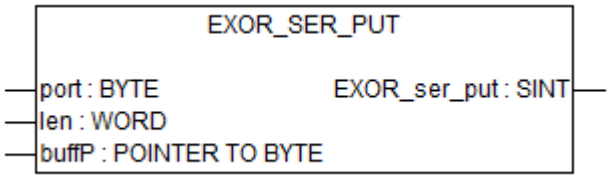
Description: result of the operation 0=no errors -1=port already in use



Name: **EXOR_ser_put**

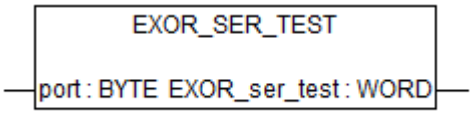
Description: write some characters to one serial port

ATTENTION: dimension of array must be large enough to contain number of characters specified in LEN parameter



Name: **EXOR_ser_test**

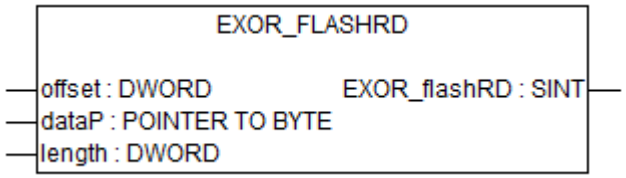
Description: test one port to know if characters are available; returns number of chars in the input queue



Library: EXOR_sys.lib

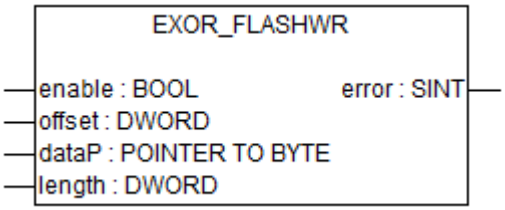
Name: **EXOR_flashRD**

Description: read one or more consecutive bytes from main flash memory



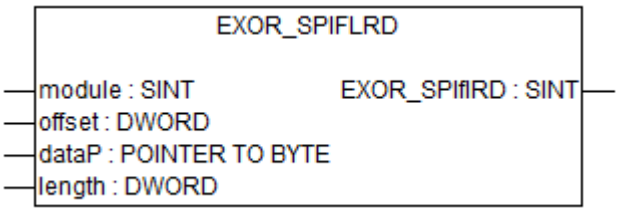
Name: **EXOR_flashWR**

Description: write one or more consecutive bytes into main flash memory (available on SCM05 - SCM11 - SCM21)



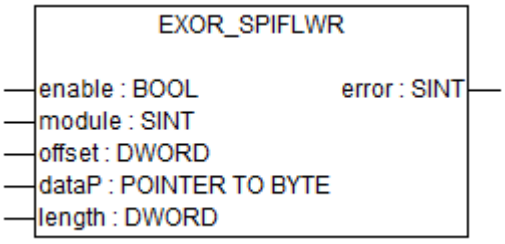
Name: **EXOR_SPIflRD**

Description: read one or more consecutive bytes from SPI flash devices on master board or slave boards through the I/O bus



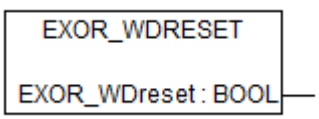
Name: **EXOR_SPIflWR**

Description: write one or more consecutive bytes into SPI flash devices on master board or slave boards through the I/O bus



Name: **EXOR_WDreset**

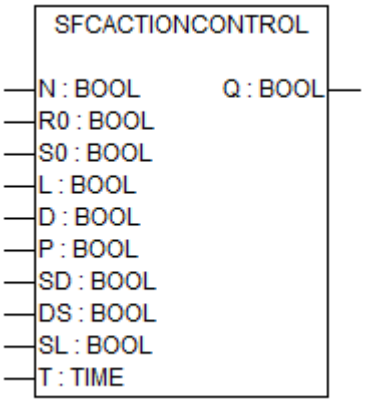
Description: clear the watch dog giving more time to CPU to run PLC program (about 1.3 seconds)



Library: Iecsfc.lib

Name: **SFCActionControl**

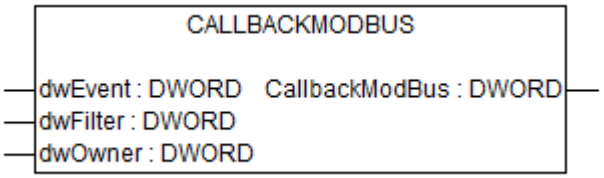
Description: Action control block for IEC 61131-3 SFC actions.



Library: Modbus_UDP.lib

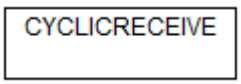
Name: **CallbackModBus**

Description:



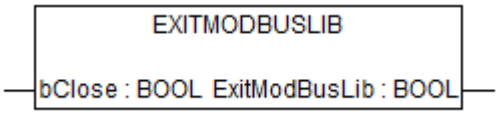
Name: **CyclicReceive**

Description: This program has to be called cyclic without arguments by the user-application.



Name: **ExitModBusLib**

Description:

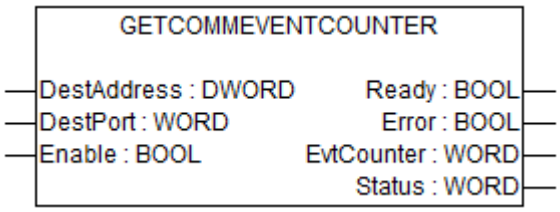


Name: **FindRegisteredQuery**

Description:

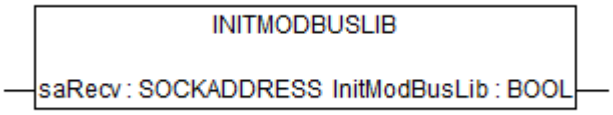
Name: **GetCommEventCounter**

Description:



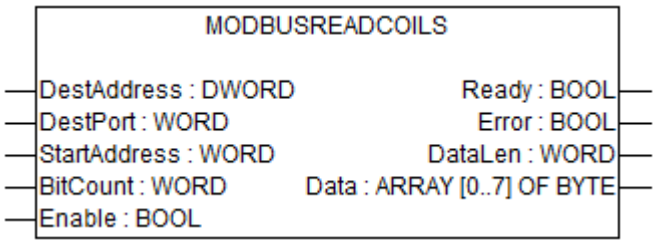
Name: **InitModBusLib**

Description:



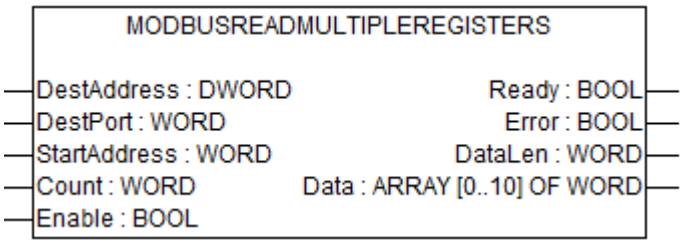
Name: **ModBusReadCoils**

Description:



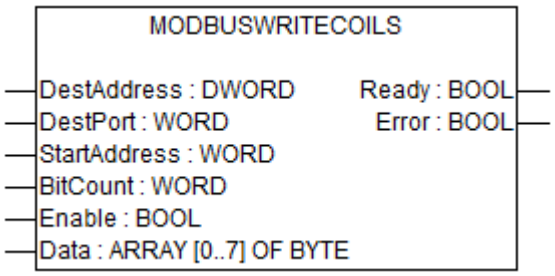
Name: **ModBusReadMultipleRegisters**

Description:



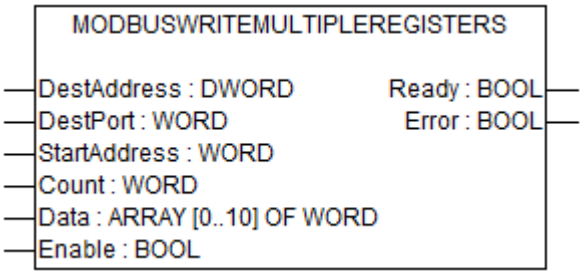
Name: **ModBusWriteCoils**

Description:



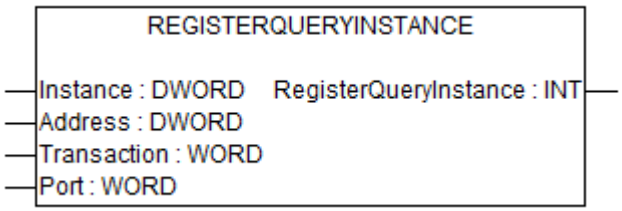
Name: **ModBusWriteMultipleRegisters**

Description:



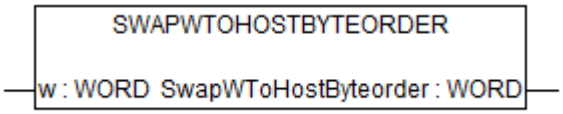
Name: **RegisterQueryInstance**

Description:



Name: **SwapWToHostByteorder**

Description:

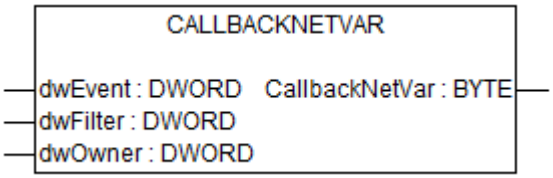


Library: NetVarUdp_LIB_V23.lib

➢ NetVar POUs

Name: **CallbackNetVar**

Description:

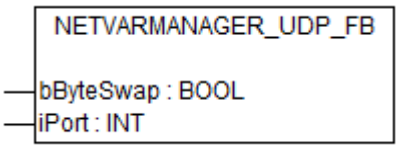


Name: **NetVarGetConnectStatus_UDP**

Description:

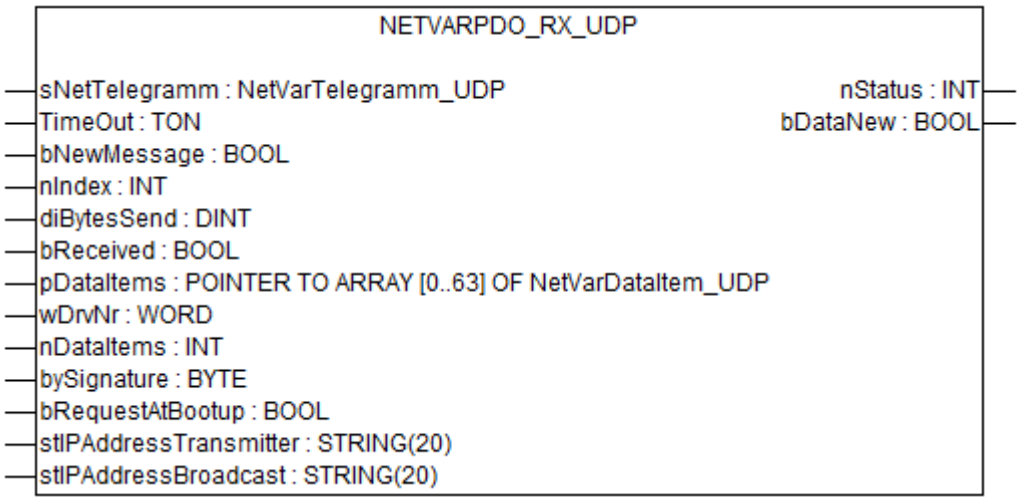
Name: **NetVarManager_Udp_FB**

Description:



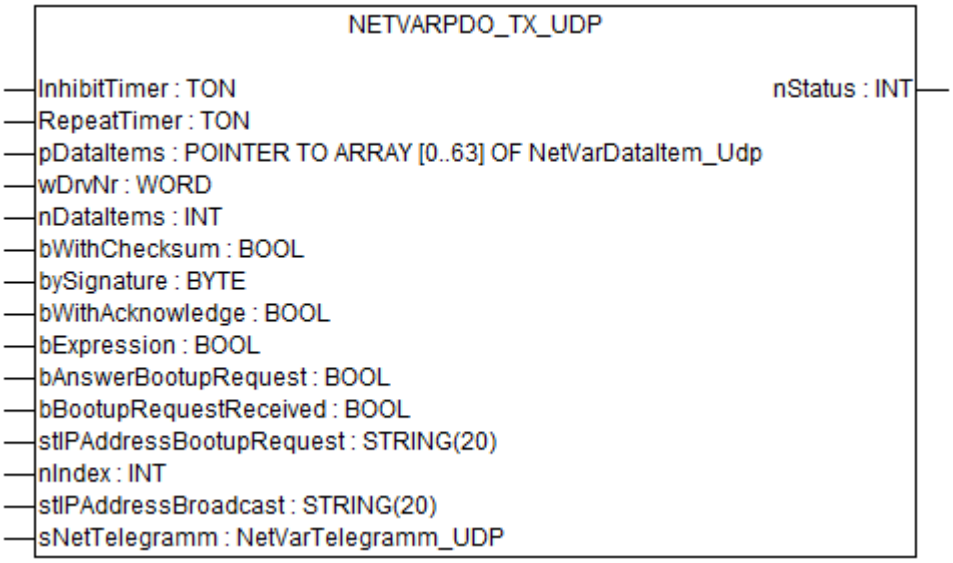
Name: **NetVarPDO_Rx_Udp**

Description:



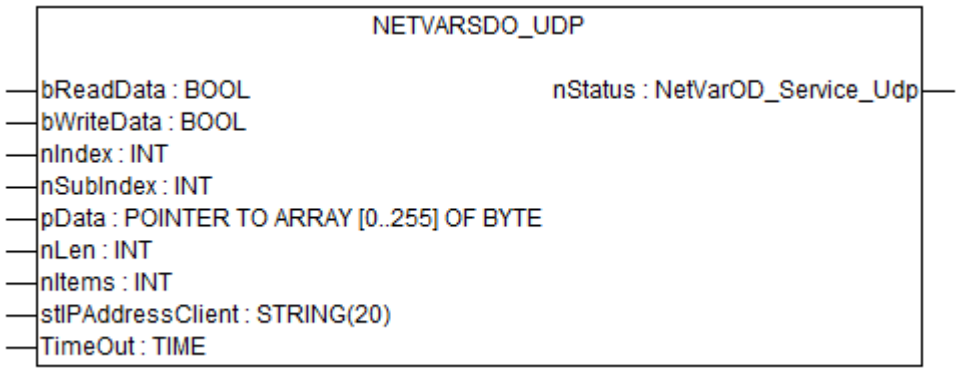
Name: **NetVarPDO_Tx_Udp**

Description:



Name: **NetVarSDO_Udp**

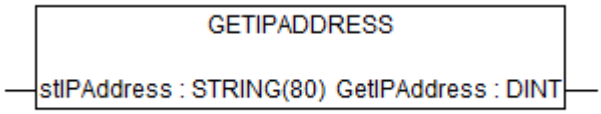
Description:



➢ Udp specific

Name: **GetIPAddress**

Description:

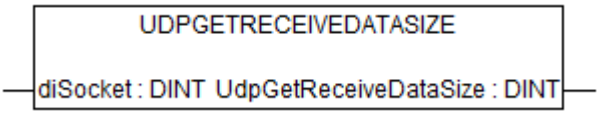


Name: **UdpGetReceiveDataSize**

Description: Function to receive data via UDP protocol.

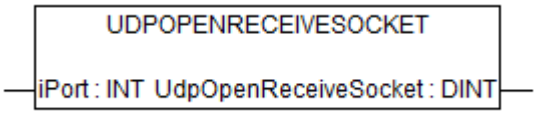
Return: diBytesReceived:Number of bytes received

stIPAddressTransmitter: IP-address of client, that sent the data



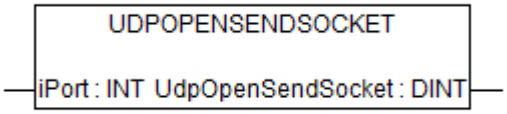
Name: **UdpOpenReceiveSocket**

Description:



Name: **UdpOpenSendSocket**

Description:

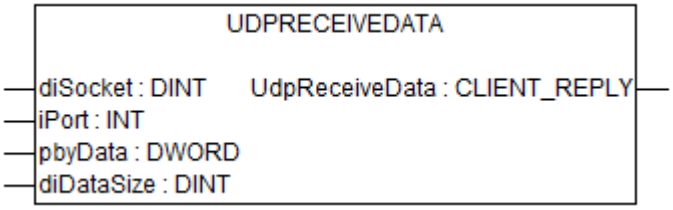


Name: **UdpReceiveData**

Description: Function to receive data via UDP protocol.

Return: diBytesReceived:Number of bytes received

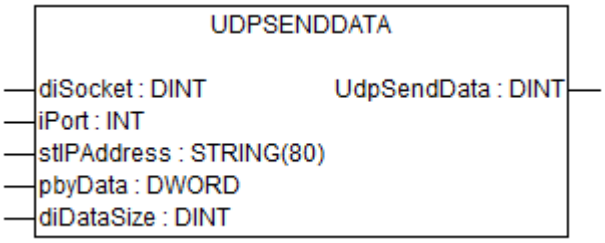
stIPAddressTransmitter: IP-address of client, that sent the data



Name: **UdpSendData**

Description: Function to send data via UDP protocol.

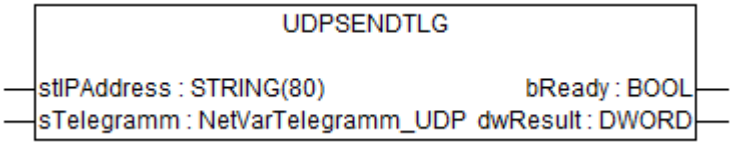
Return: Number of bytes really sent



Name: **UdpSendTlg**

Description: Function to send data via UDP protocol.

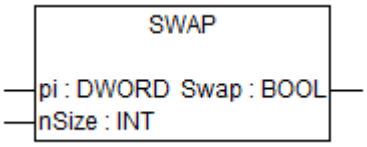
Return: Number of bytes really sent



➢ Others

Name: **Swap**

Description:

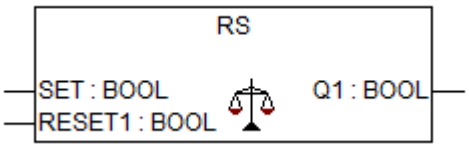


Library: Standard_V2.lib

➢ Bistable function blocks

Name: **RS**

Description: bistable function, reset dominant Q1 = NOT R1 AND (SET OR Q1)



Name: **SEMA**

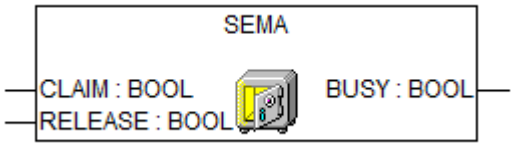
Description: Software semaphore. Interuptable!

BUSY is TRUE, if there was a call with CLAIM = TRUE,

but no call with RELEASE = TRUE.

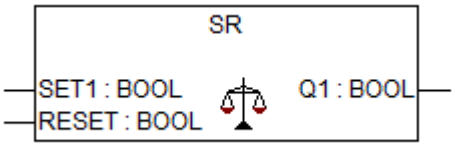
CLAIM = TRUE sets BUSY = TRUE;

REALEASE = TRUE sets BUSY = FALSE;



Name: **SR**

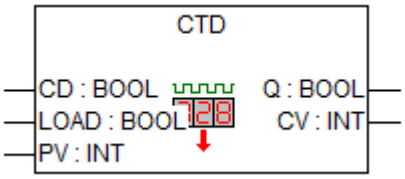
Description: bistable function, set dominant Q1 = S1 OR (NOT RESET AND Q1)



➢ Counter

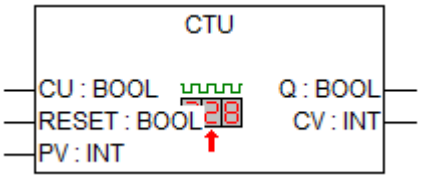
Name: **CTD**

Description: Counter Down. CV is decremented by 1 if CD has a rising edge. Q is TRUE, if CV reached 0.



Name: **CTU**

Description: Counter Up. CV is incremented by 1 if CU has a rising edge. Q is TRUE, if CV is reached PV.



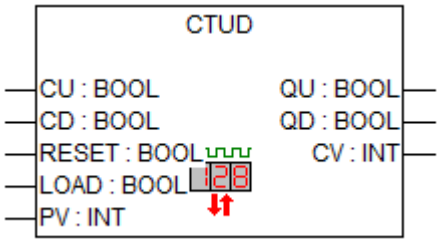
Name: **CTUD**

Description: CounTer Up Down CV is decremented by 1 if CD has a rising edge.

CV is incremented by 1 if CU has a rising edge.

QV is TRUE, if counter is 0.

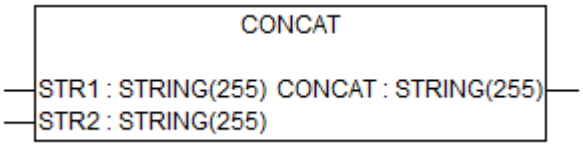
QU is TRUE, if counter is PV.



➢ String functions

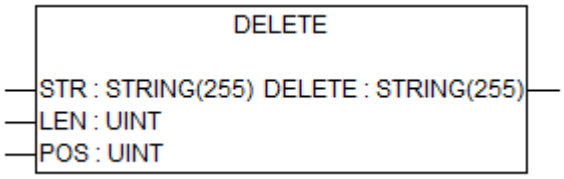
Name: **CONCAT**

Description: Concatenation of two strings



Name: **DELETE**

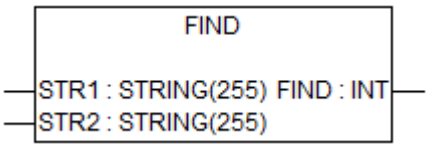
Description: Delete LEN characters of STR, beginning at the POS-th character position



Name: **FIND**

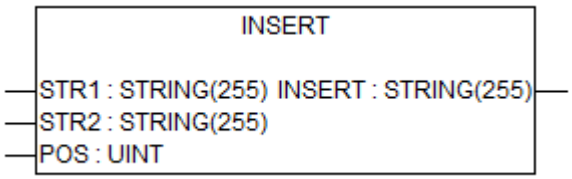
Description: Find the character position of the beginning of the first occurence of STR2 in STR1.

If no occurence of STR1 is found, then the result is 0.



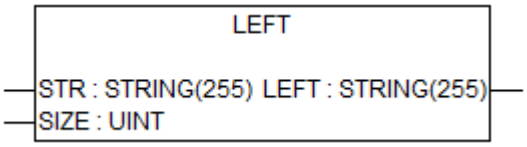
Name: **INSERT**

Description: Insert STR2 into STR1 after the POS-th character position



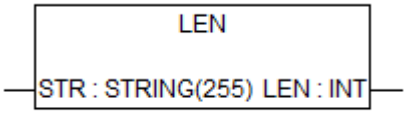
Name: **LEFT**

Description: Leftmost SIZE characters of STR



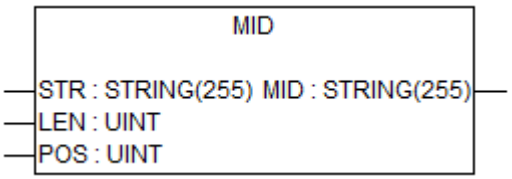
Name: **LEN**

Description: String length function



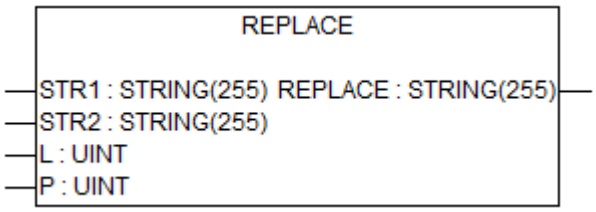
Name: **MID**

Description: LEN characters of STR, beginning at the POS-th character position



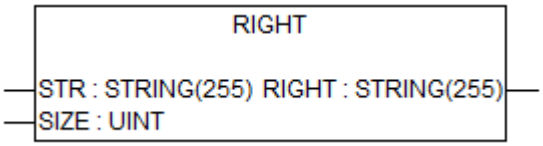
Name: **REPLACE**

Description: Replaces L characters of STR1 by STR2, starting at the POS-th character position



Name: **RIGHT**

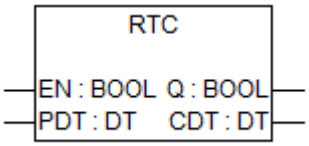
Description: Rightmost SIZE characters of STR



➢ Timer

Name: **RTC**

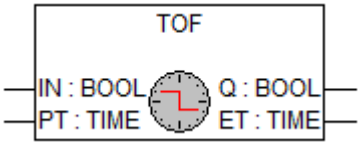
Description:



Name: **TOF**

Description: Timer of delay.

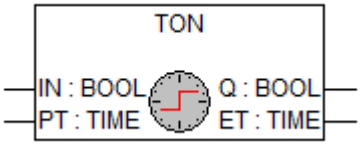
Q is FALSE, PT milliseconds after IN had a falling edge



Name: **TON**

Description: Timer on delay.

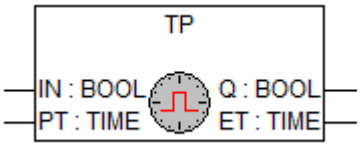
Q is TRUE, PT milliseconds after IN had a rising edge



Name: **TP**

Description: Timer Pulse

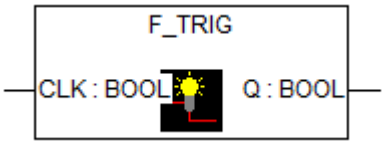
Q0 produces a High-Signal with the length of PT on every rising edge on IN



➢ Trigger

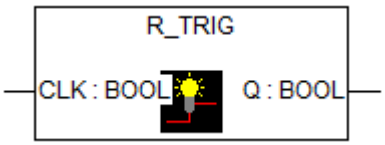
Name: **F_TRIG**

Description: Falling Edge detection



Name: **R_TRIG**

Description: Rising Edge detection



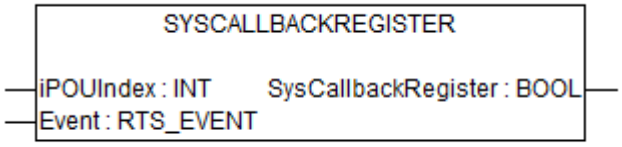
Library: SysLibCallback.lib

Name: **SysCallbackRegister**

Description:

Function to register a definite callback function on runtime events.

Return TRUE, if callback function can be registered.

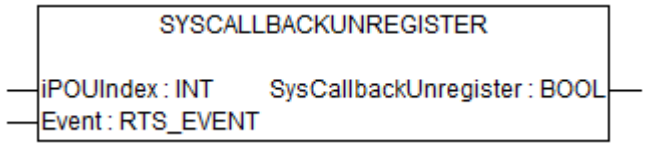


Name: **SysCallbackUnregister**

Description:

Function to unregister a definite callback function on runtime events.

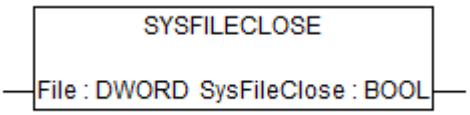
Return TRUE, if callback function can be unregistered.



Library: SysLibFile.lib

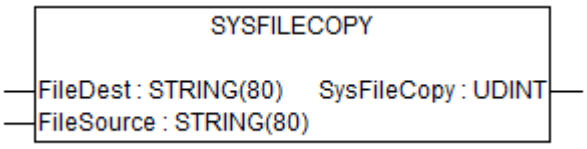
Name: **SysFileClose**

Description:



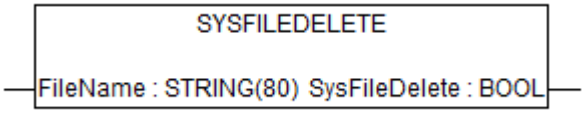
Name: **SysFileCopy**

Description:



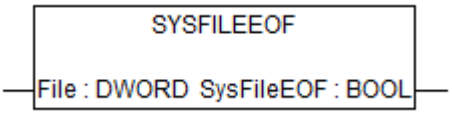
Name: **SysFileDelete**

Description:



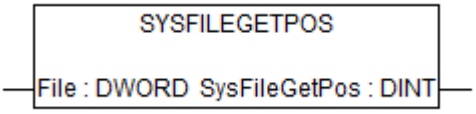
Name: **SysFileEOF**

Description:

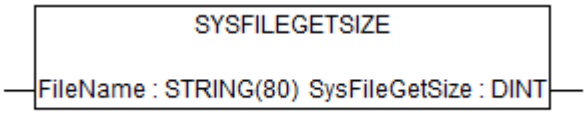


Name: **SysFileGetPos**

Description:

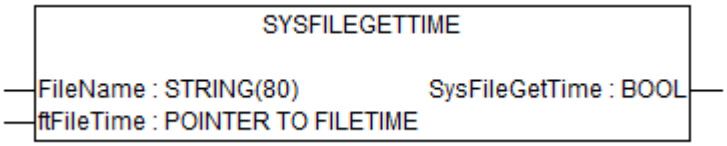


Name: **SysFileGetSize**



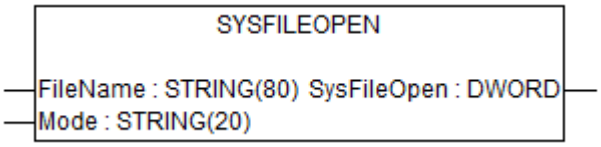
Name: **SysFileGetTime**

Description:



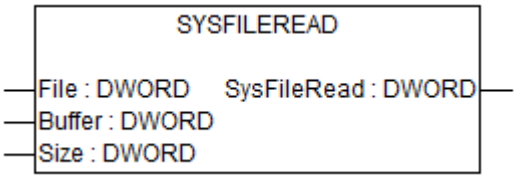
Name: **SysFileOpen**

Description:



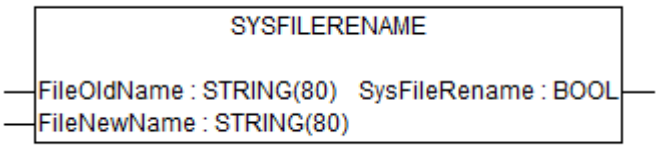
Name: **SysFileRead**

Description:

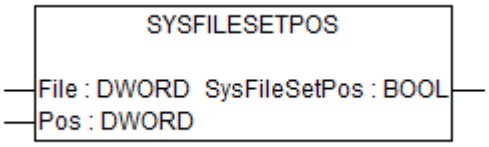


Name: **SysFileRename**

Description:

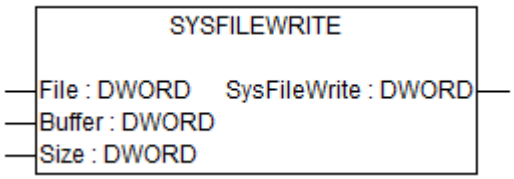


Description:



Name: **SysFileWrite**

Description:

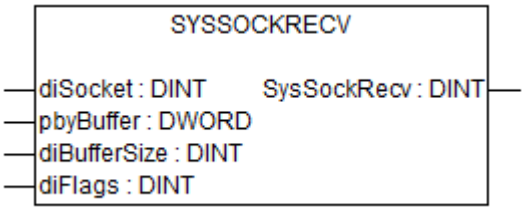


Library: SysLibSocket.lib

➢ TCP specific

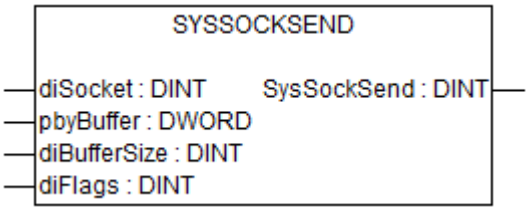
Name: **SysSockRecv**

Description:



Name: **SysSockSend**

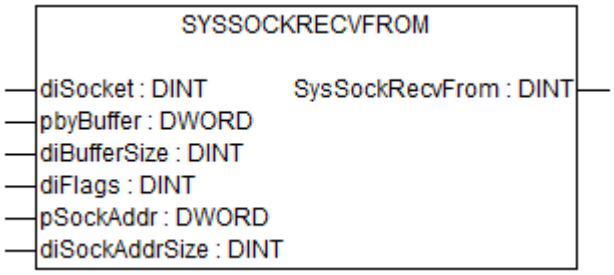
Description:



➢ UDP specific

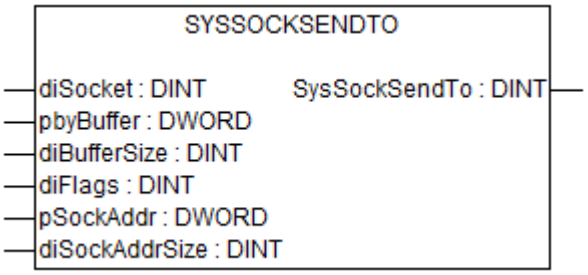
Name: **SysSockRecvFrom**

Description:



Name: **SysSockSendTo**

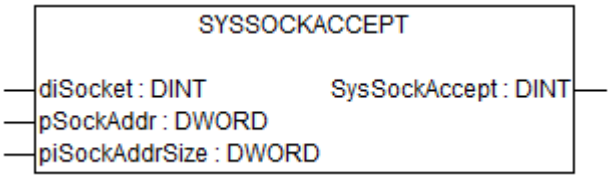
Description:



➢ others

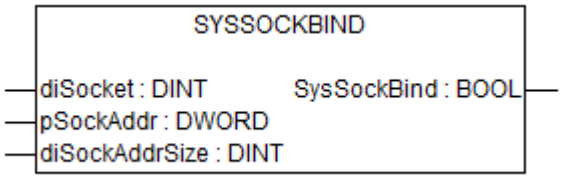
Name: **SysSockAccept**

Description:



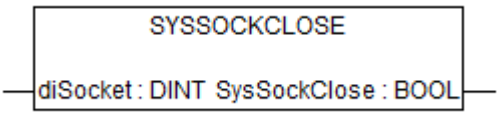
Name: **SysSockBind**

Description:



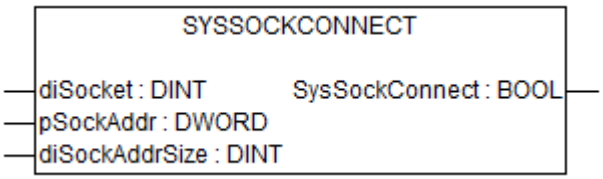
Name: **SysSockClose**

Description:



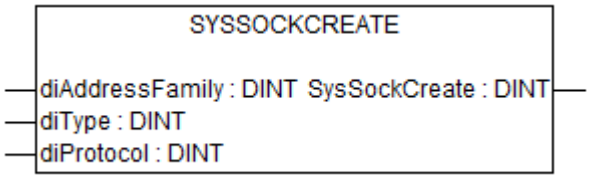
Name: **SysSockConnect**

Description:



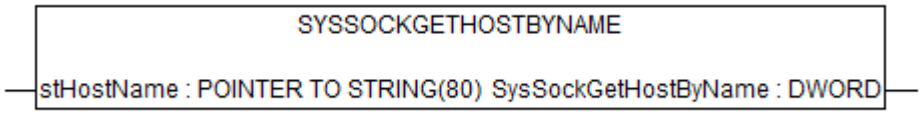
Name: **SysSockCreate**

Description:



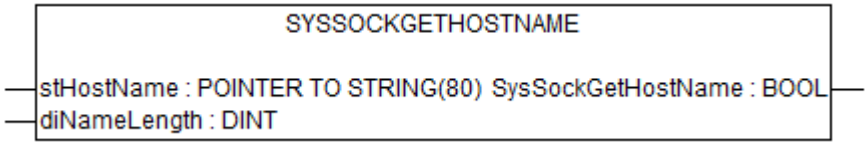
Name: **SysSockGetHostByName**

Description:



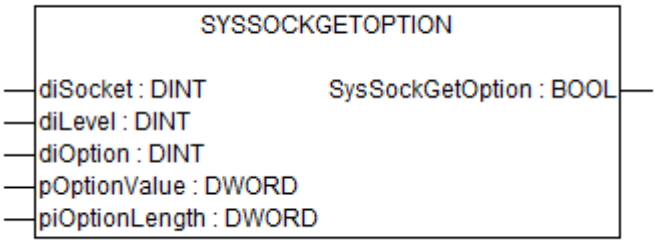
Name: **SysSockGetHostName**

Description:



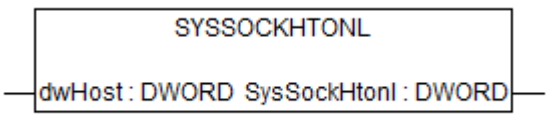
Name: **SysSockGetOption**

Description:



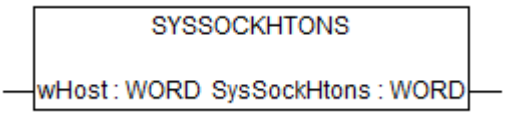
Name: **SysSockHtonl**

Description:



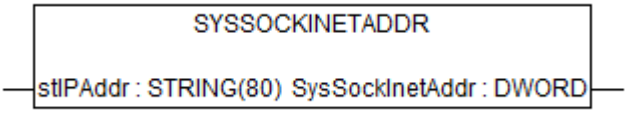
Name: **SysSockHtons**

Description:



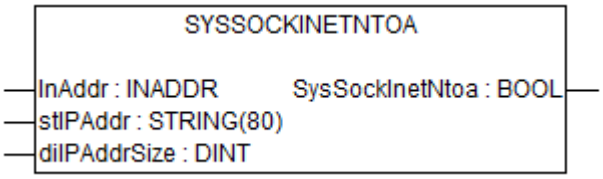
Name: **SysSockInetAddr**

Description:



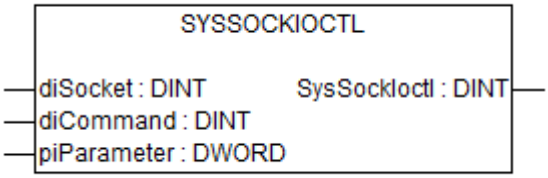
Name: **SysSockInetNtoa**

Description:



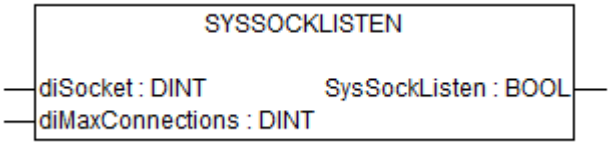
Name: **SysSockIoctl**

Description:



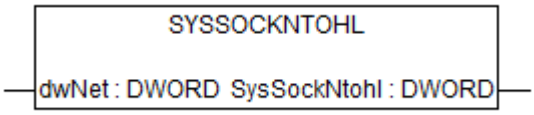
Name: **SysSockListen**

Description:



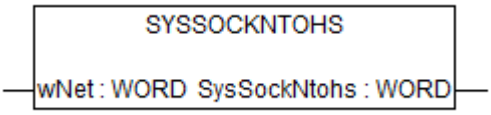
Name: **SysSockNtohl**

Description:



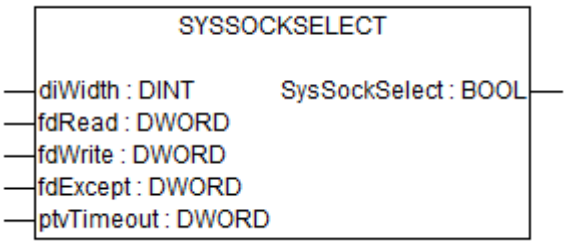
Name: **SysSockNtohs**

Description:



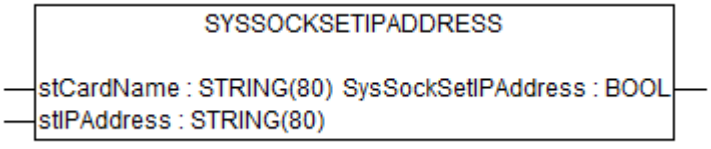
Name: **SysSockSelect**

Description:



Name: **SysSockSetIPAddress**

Description:

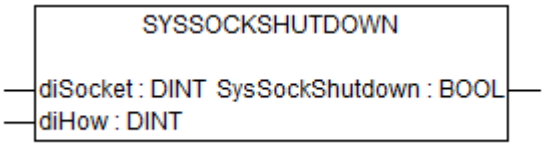


Name: **SysSockSetOption**

Description:

Name: **SysSockShutdown**

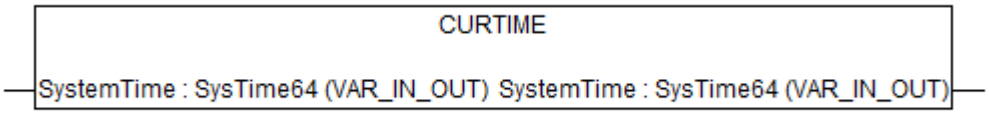
Description:



Library: SysLibTime.lib

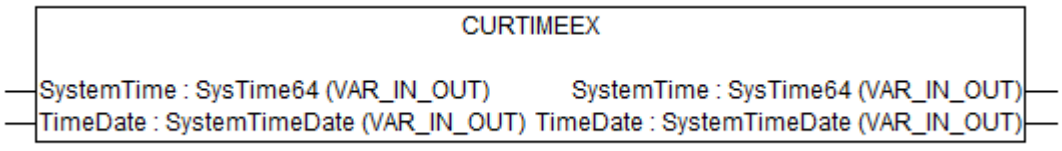
Name: **CurTime**

Description:



Name: **CurTimeEx**

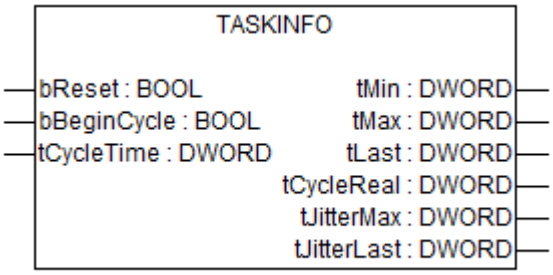
Description:



Library: SysTaskInfo.lib

Name: **taskinfo**

Description:



Library: TCPUdpLib.lib

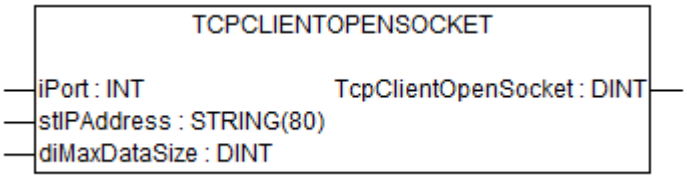
➢ TCPSpecific

Name: **TcpClientOpenSocket**

Description:

Open TCP client socket to connect to server.

Return: Socket-Id for connection session

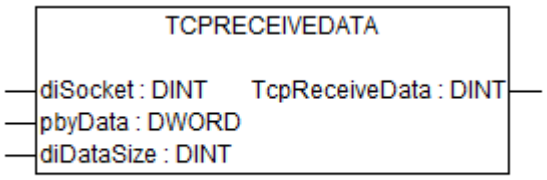


Name: **TcpReceiveData**

Description:

Receive data from a tcp socket.

Return: Number of bytes received

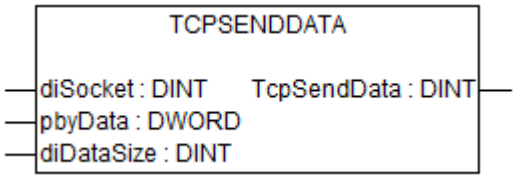


Name: **TcpSendData**

Description:

Send data via TCP socket.

Return: Number of bytes sent

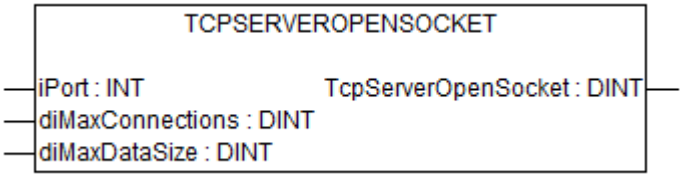


Name: **TcpServerOpenSocket**

Description:

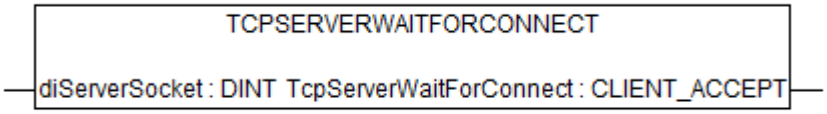
Open a TCP server socket.

Return: Main Socket Id



Name: **TcpServerWaitForConnect**

Description: Wait for a TCP client to connect.



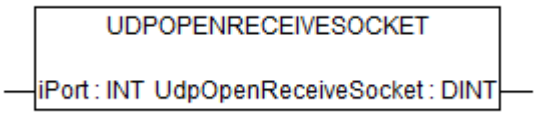
- **Udp specific**

Name: **UdpOpenReceiveSocket**

Description:

Open socket to receive data via UDP.

Return: Socket Id

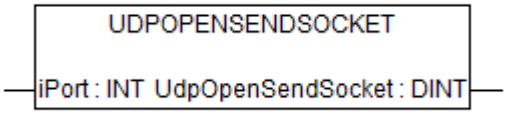


Name: **UdpOpenSendSocket**

Description:

Open socket to send data via UDP.

Return: Socket Id



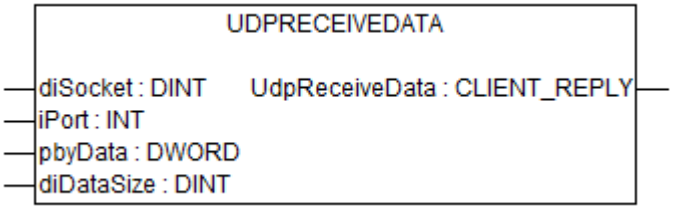
Name: **UdpReceiveData**

Description:

Function to receive data via UDP protocol.

Return: diBytesReceived:Number of bytes received

stIPAddressTransmitter: IP-address of client, that sent the data

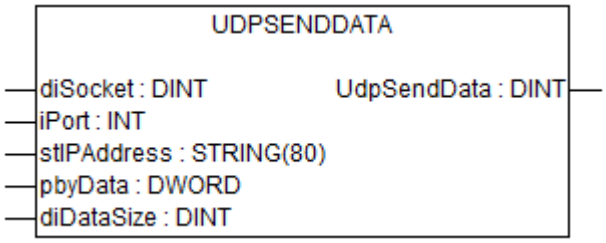


Name: **UdpSendData**

Description:

Function to send data via UDP protocol.

Return: Number of bytes realy sent



➢ others

Name: **ConvertPAddressToString**

Description:

Convert the IP-Address to a string in dot-notation, e.g. '192.168.100.2' .

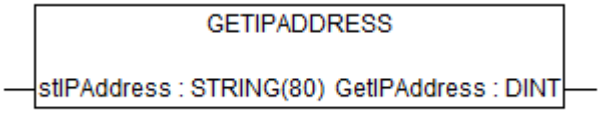
Return: IP-address

Name: **GetIPAddress**

Description:

Convert IP-Address from string to address.

Return: IP-address

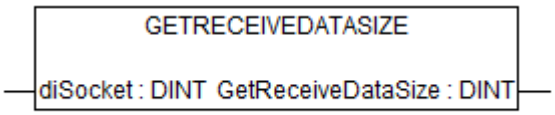


Name: **GetReceiveDataSize**

Description:

Function to get received number of bytes in socket, without removing it from the socket.

Return: Number of bytes received

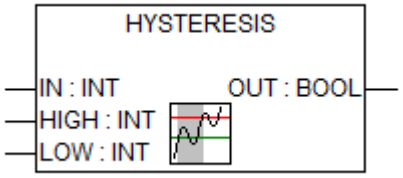


Library: Util.lib

➢ Analog monitors

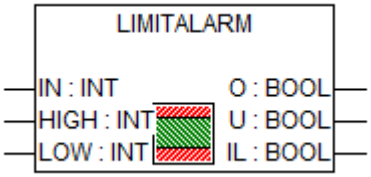
Name: **HYSTERESIS**

Description: function block, which sets OUT to TRUE, if input IN is less than LOW, and which sets OUT to FALSE, if input IN is bigger than HIGH



Name: **LIMITALARM**

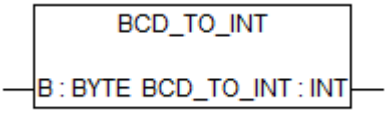
Description: function block monitoring, whether the input value IN lies between LOW and HIGH



➢ BDC conversion

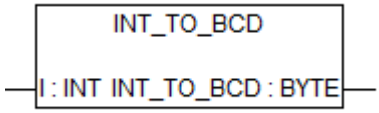
Name: **BCD_TO_INT**

Description: function reads BCD-Byte and returns its INT value; a wrong BCD value causes a return of -1



Name: **INT_TO_BCD**

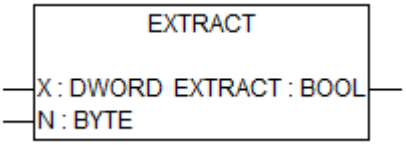
Description: function reads INT and returns its BCD value; a wrong INT value (<0 or >99) causes a return of 255



➢ Bit/byte functions

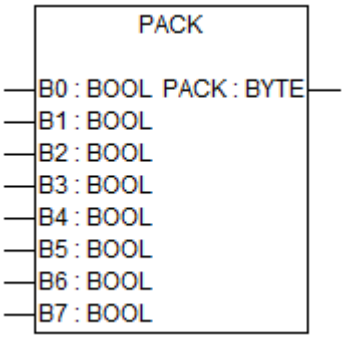
Name: **EXTRACT**

Description: function returns Bit N of value X



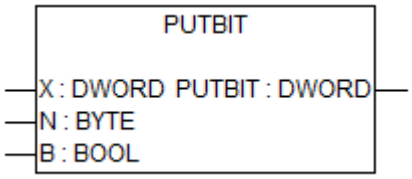
Name: **PACK**

Description: pack eight bits into 1 byte



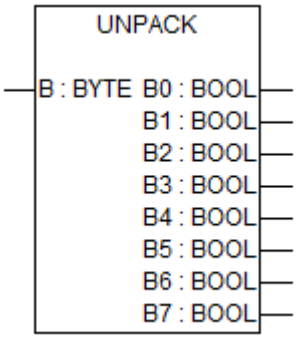
Name: **PUTBIT**

Description: function sets bit N of X to B and returns new value



Name: **UNPACK**

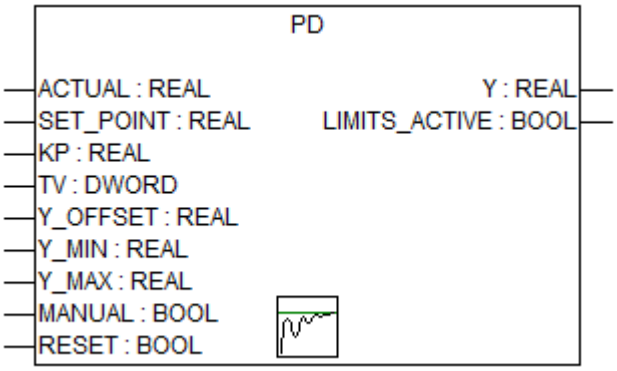
Description: function block unpacks byte into 8 bits



➢ Controller

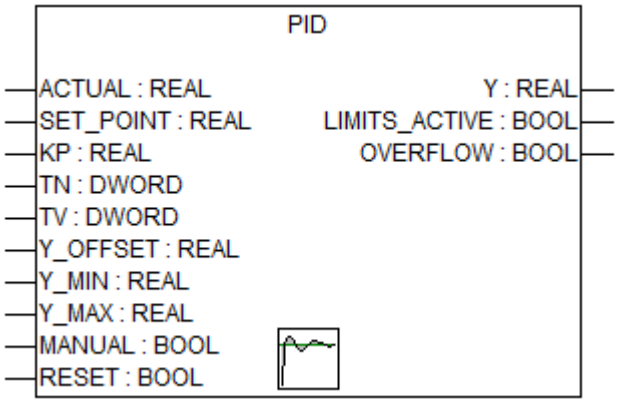
Name: **PD**

Description: PD controller



Name: **PID**

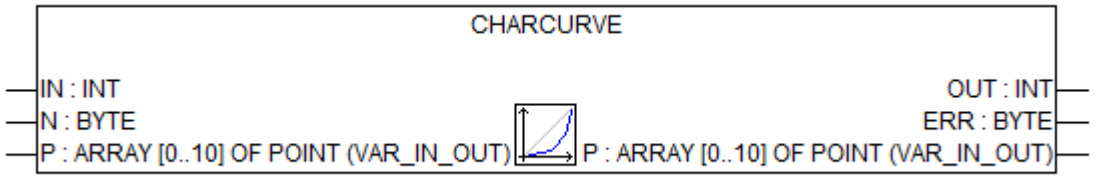
Description: PD controller



➢ Function manipulators

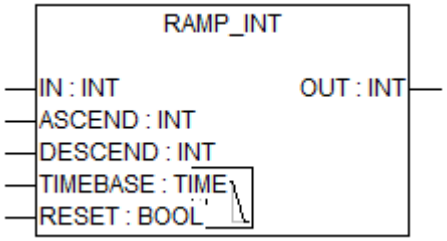
Name: **CHARCURVE**

Description: function block to map an input signal on a characteristic curve



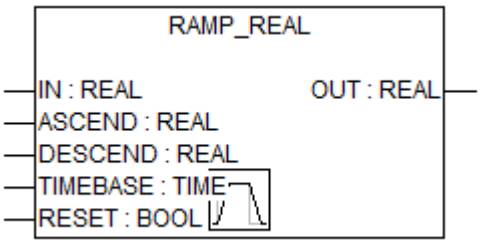
Name: **RAMP_INT**

Description: function block to limit the slope of a value to a certain value



Name: **RAMP_REAL**

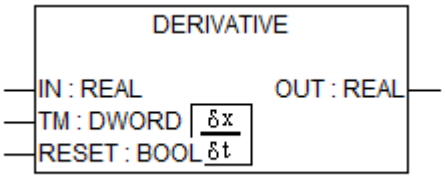
Description: function block to limit the slope of a value to a certain value



➢ Mathematical functions

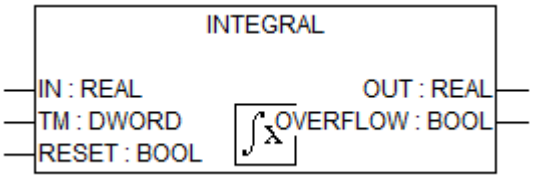
Name: **DERIVATIVE**

Description: function block to approximate the derivative of a value



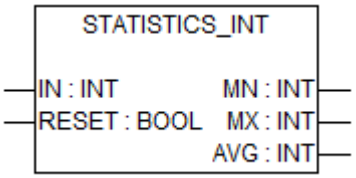
Name: **INTEGRAL**

Description: function block to approximate the integral of IN over the time



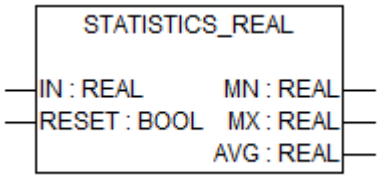
Name: **STATISTICS_INT**

Description: function block for statistics: calculating minimum, maximum and average of an input value



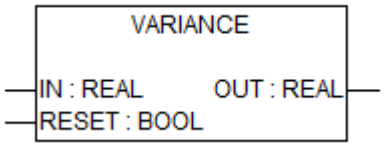
Name: **STATISTICS_REAL**

Description: function block for statistics: calculating minimum, maximum and average of an input value



Name: **VARIANCE**

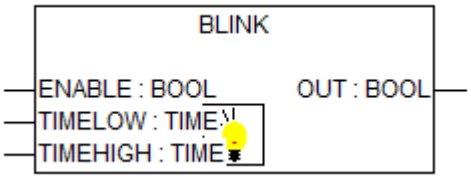
Description: function block to calculate the mathematical variance of a variable



➢ Signal generators

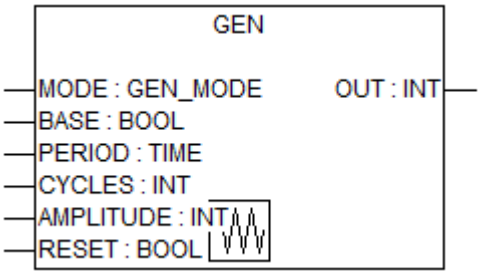
Name: **BLINK**

Description: Function block to simulate a turn signal



Name: **GEN**

Description: function block for generating some periodic functions

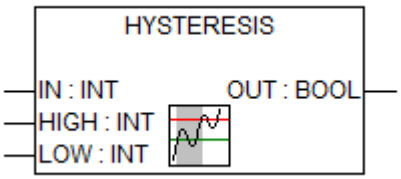


Library: Util_no_Real.lib

➢ Analog monitors

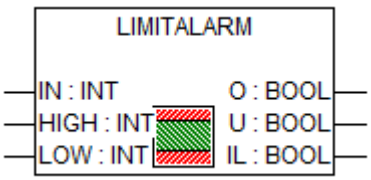
Name: **HYSTERESIS**

Description: function block, which sets OUT to TRUE, if input IN is less than LOW, and which sets OUT to FALSE, if input IN is bigger than HIGH.



Name: **LIMITALARM**

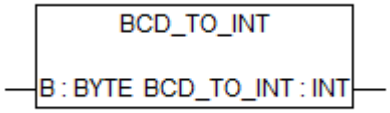
Description: function block monitoring, whether the input value IN lies between LOW and HIGH



➢ BCD conversion

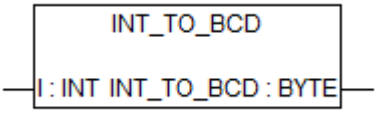
Name: **BCD_TO_INT**

Description: function reads BCD-Byte and returns its INT value; a wrong BCD value causes a return of -1



Name: **INT_TO_BCD**

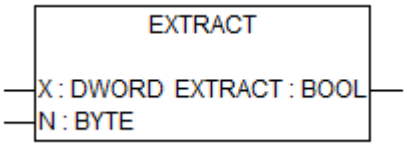
Description: function reads INT and returns its BCD value; a wrong INT value (<0 or >99) causes a return of 255



➢ Bit/byte functions

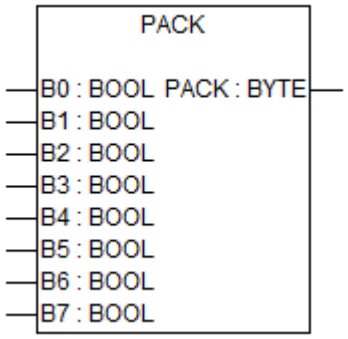
Name: **EXTRACT**

Description: function returns Bit N of value X



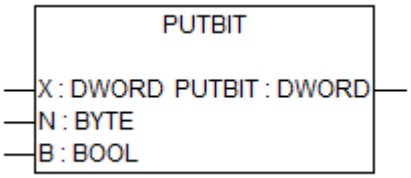
Name: **PACK**

Description: pack eight bits into 1 byte



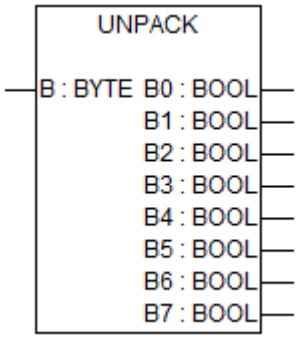
Name: **PUTBIT**

Description: function sets bit N of X to B and returns new value



Name: **UNPACK**

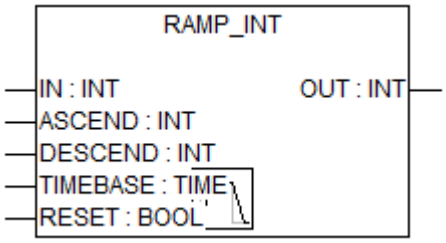
Description: function block unpacks byte into 8 bits



➢ Function manipulators

Name: **RAMP_INT**

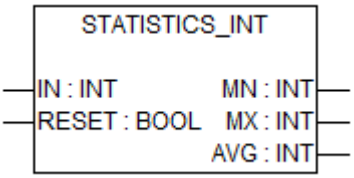
Description: function block to limit the slope of a value to a certain value



➢ Mathematical functions

Name: **STATISTICS_INT**

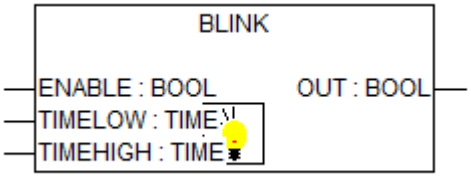
Description: function block for statistics: calculating minimum, maximum and average of an input value



➢ Signal generators

Name: **BLINK**

Description: Function block to simulate a turn signal



Appendix C: CANopen I/O with ISAGRAF PLC

This section describes the use of CANOpen in the ISaGRAF environment.

Isagraf Library

The use of CANopen in ISAGRAF requires loading library for CANopen. In particular, it is necessary to load the library for the IO boards canocfg2, canodi, canodo and canao:

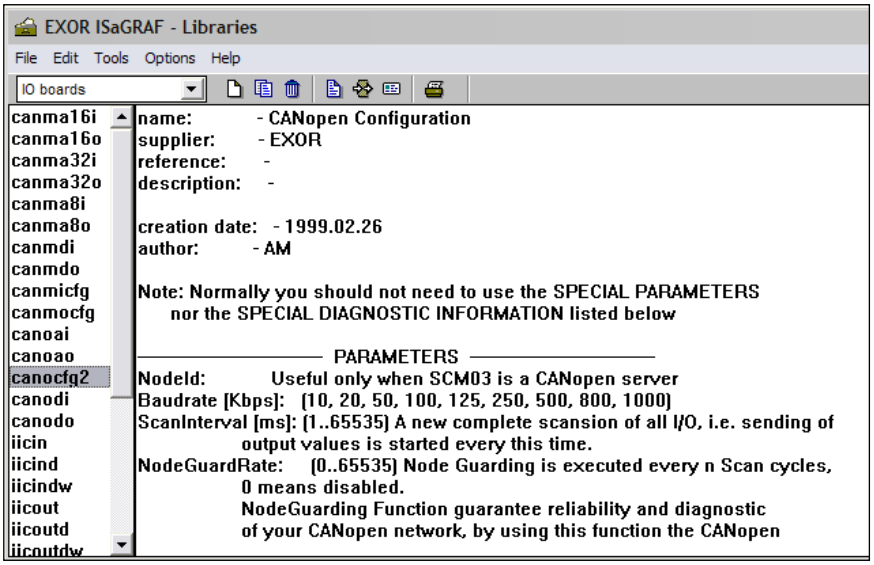


Figure 67 – Isagraf: libraries

Boards canocfg2

The board canocfg2 is used to configure the communication via the TWS CANopen master. The integration of the board is done through the menu Project | IO Connection.

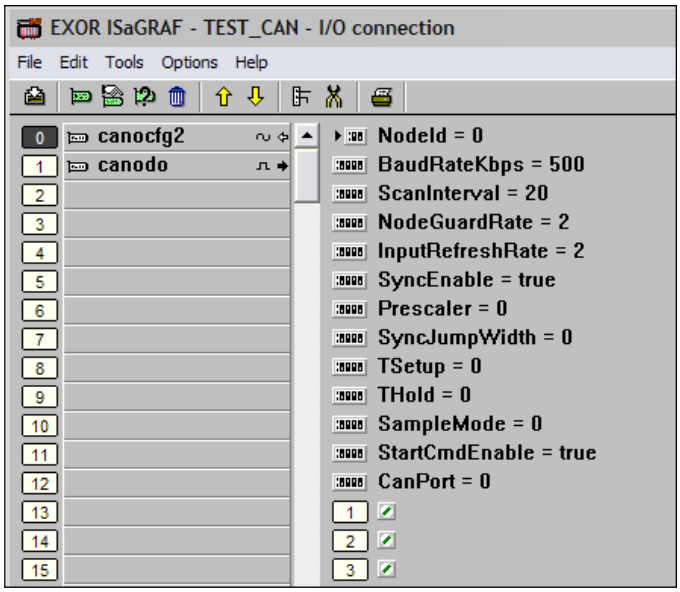


Figure 68 - Isagraf: I/O connection

The fields of the IO Boards have the following meanings:
• Nodeid: identification of the CANOpen master node
• Baudrate: Select the speed of the network CANOpen. It must be the same as set by the DIP switch on the slave.
• ScanInterval: scan time of the inputs and outputs in milliseconds.
• NodeGuardRate: number of cycles to scan before sending the NodeGuard message. The value 0 means disabled.
• InputRefreshRate: request for inputs delivery through Remote Transmission Request (not supported by Z-PC CANopen slaves)
• SyncEnable: Enabling SYNC message from the master
• StartCmdEnable: If TRUE the NMT start network command is given at boot-up or in case of Node Guarding error .

The special parameters (prescaler (1 .. 64), SyncJumpWidth (1 .. 4), TSetup (1 .. 16) THold (1 .. 8), SampleMode (0 .. 1)) are used to select the speed of the network and have special meaning only if the Baudrate parameter is equal to zero. Normally should not be used.

Now you can point directly to the inputs and outputs of the slave stations using the appropriate boards. The example below shows the use of a cando to point to the first digital output of the slave with station number 1.

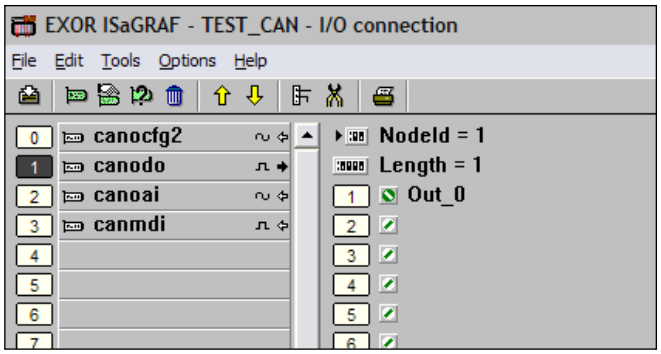


Figure 69 - Isagraf: an example of I/O connection

Appendix D: Library “Controlli” for Microsoft® Visual Studio .NET

Seneca provides a free library for Visual Studio .NET, that can be used from the version 2005. This library is free downloadable from the web site www.seneca.it. This library allows the communication through serial port of a PC to the I/O Modbus.

NOTE: Seneca provides a variety of RS485-USB converters, for example S117P and S117P1; these devices allows to connect the Seneca Z-PC Line modules with the PC.

Public Sub **Serial_Open**(ByVal *comName* As **String,** ByVal *baudRate* As **Integer,** ByVal *parity* As **System**.**IO**.**Ports**.**Parity,** ByVal *nbit* As **Integer,** ByVal *stopBits* As **System**.**IO**.**Ports**.**StopBits,** ByVal *timeout* As **Integer,** ByVal *flowControl* As **System**.**IO**.**Ports**.**Handshake**)

Opening of the serial port

Public Sub **Modbus_setSerial**(ByRef *com* As **Controlli**.**clsSeriale**)

Member of: **Controlli**.**clsModbus**

Description:

Set the serial interface to be used

Parameters:

com: Serial port interface

Public Property **nRetry**() As **Integer**

Member: **Controlli**.**clsModbus**

Description:

Set number of retries of Modbus request

Return values:

Actual number of retries

Note:

Must be from 1 to 9 included

Public Property **Address**() As **Integer**

Member of: **Controlli**.**clsModbus**

Description:

Set the Modbus address

Return values:

Actual Modbus address

Note:

Must be from 1 to 253 included

Public Function **Read_Coils**(ByVal *startRegister* As **Integer,** ByVal *nCoils* As **Integer**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Read Coils Modbus request

Parameters:

startRegister: Start register address (0 to 65535)

nCoils: Number of coils (1 to 2000)

Return values:

Coils value

Public Function **Read_Discrete_Inputs**(ByVal *startRegister* As **Integer,** ByVal *nInputs* As **Integer**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Read Discrete Inputs Modbus request

Parameters:

startRegister: Start register address (0 to 65535)

nInputs: Number of discrete inputs (1 to 2000)

Return values:

Discrete inputs

Note:

Public Function **Read_File_Record_Simple**(ByVal *FileNumber* As **Integer,** ByVal *RecordNumber* As **Integer,** ByVal *Nrecord* As **Integer**) As **Integer**()

Member of: **Controlli**.**clsModbus**

Description:

Read File Record Modbus request

Parameters:

FileNumber: File number (1 to 65535)

RecordNumber: Record number (0 to 9999)

Nrecord: Number of records

Return values:

File/Record values

Public Function **Read_Holding_Registers**(ByVal *startRegister* As **Integer,** ByVal *nRegister* As **Integer**) As **Integer**()

Member of: **Controlli**.**clsModbus**

Description:

Read Holding Registers Modbus request

Parameters:

startRegister: Start register address (0 to 65535)

nRegister: Number of registers to read (1 to 125)

Return values:

Registers value

Note:

Public Function **Read_Input_Registers**(ByVal *startRegister* As **Integer,** ByVal *nRegister* As **Integer**) As **Integer**()

Member of: **Controlli**.**clsModbus**

Description:

Read Input Registers Modbus request

startRegister: Start register address (0 to 65535)

nRegister: Number of input registers (1 to 125)

Return values:

Input registers

Public Property **Timeout**() As **Integer**

Member of: **Controlli**.**clsModbus**

Description:

Set the timeout of the Modbus request

Return values:

Actual timeout as ms

Note:

Must be > 0

Public Function **Write_Multiple_Coils**(ByVal *startRegister* As **Integer,** ByVal *nCoils* As **Integer,** ByRef *Values*() As **Byte**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Write Multiple Coils Modbus request

Parameters:

startRegister: Start register address (0 to 65535)

nCoils: Number of coils to write (0 to 1968)

Values: Coils values

Public Function **Write_Multiple_Register**(ByVal *startRegister* As **Integer,** ByVal *nRegister* As **Integer,** ByRef *Values*() As **Integer**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Write Multiple Registers Modbus request

Parameters:

startRegister: Start register address (0 to 65535)

nRegister: Number of registers to write (1 to 123)

Values: Values to write

Return values:

Modbus reply

Public Function **Write_Single_Coil**(ByVal *Register* As **Integer,** ByVal *Value* As **Integer**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Write Single Coil Modbus request

Parameters:

Register: Register address (0 to 65535)

Value: Coil value (0 or 65280 (0xFF00))

Return values:

Modbus reply

Public Function **Write_Single_Register**(ByVal *Register* As **Integer,** ByVal *Value* As **Integer**) As **Byte**()

Member of: **Controlli**.**clsModbus**

Description:

Write Single Register Modbus request

Parameters:

Register: Register address (0 to 65535)

Value: Value to write

Return values:

Modbus reply

Example:

```
imports controlli

            If Me.mCom.Serial_isOpen Then
                Me.mCom.Serial_Close()
            End If
```

```
            Me.mCom.Serial_Open("com1", 38400,parity.none, 8,
StopBits.One, 300, Handshake.None)
            Me.mRtu.Modbus_setSerial(Me.mCom)
            Me.mRtu.nRetry = 3
            Me.mRtu.Address = 1
            Me.mRtu.Timeout = 500

            Dim values(1) As Integer
            Dim reply(-1) As Byte

            Reading of two registers: 40001 e 40002
            reply = mRtu.Read_Holding_Registers(0, 2)

            If reply.Length = 0 Then
                sendModbusError(Nothing, sh)
                Return False
            End If

'In reply(0) there is the register 40001
'In reply(1) there is the register 40002
```